REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATE COVERED
	4 May 2007	

4. TITLE AND SUBTITLE
Swarm Manipulation of Large Surface Vessels

5. FUNDING NUMBERS

6. AUTHOR(S)
Smith, Erik T.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)		8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSORING/MONITORING AGENCY REPORT NUMBER
US Naval Academy Annapolis, MD 21402		Trident Scholar project report no. 359 (2007)

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT	12b. DISTRIBUTION CODE
This document has been approved for public release; its distribution is UNLIMITED.	

13. ABSTRACT The goal of this Trident project was to develop an independent control scheme to allow a team of autonomous tugboats to move a large disabled vessel, such as a barge, to a desired position and orientation. Independence refers to the extent to which each tugboat's actions were free from knowledge of the locations and actions of other tugboats. Performance of the team was quantified by measuring the positional error and time required to affect the motion, while respecting maximum power constraints on the thrust. Applications of the project include difficult or dangerous tasks such as moving disabled vessels or vessels "not under command" through hostile or dangerous areas, and transportation of large objects such as marine construction equipment, off-shore bases, drilling platforms, and sonar arrays. Although it would be ideal to increase both the independence and performance of the system, it must be realized that by increasing one of these, the other is typically degraded. In order to measure performance, a control strategy (the baseline) was designed that required the attachment points of all tugboats to be known. However, this architecture was not desirable, since it was less independent of system knowledge. In contrast, to allow for the elimination of known tugboat location, an adaptive control strategy was developed which resulted in degradation of performance. These two Scenarios were explored and in the course of solving them, the tradeoff between performance and independence was quantified. To the author's knowledge, this is the first study of its kind and complexity.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Adaptive control, Swarm robotics, Parameter identification, and automatic control	140
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATE COVERED
	4 May 2007	

4. TITLE AND SUBTITLE
Swarm manipulation of large surface vessels

Smith, Erik T.

5. FUNDING NUMBERS

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)		8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSORING/MONITORING AGENCY REPORT NUMBER
US Naval Academy Annapolis, MD 21402		**Trident Scholar project report no. 359 (2007)**

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT	12b. DISTRIBUTION CODE
This document has been approved for public release; its distribution is UNLIMITED.	

13. ABSTRACT (cont.) Although previous work has studied adaptive control of a multi-input and multi-output system, its extent and focus was not close to this research. Each tugboat used on-line adaptive control methods to compensate for the unknown actions of other swarm members. The analysis was verified through simulation. In addition, an experimental proof-of-concept device was built and in-water experiments were used to validate the results. An incremental approach to experiment design was used to mitigate the challenges of in-water experimentation.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Adaptive Control, Swarm Robotics, Parameter Identification, and Automatic Control	140
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT

U.S.N.A. --- Trident Scholar project report; no. 359 (2007)

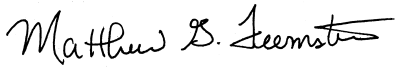
SWARM MANIPULATION OF LARGE SURFACE VESSELS

by

Midshipman 1/c Erik Thomas Smith
United States Naval Academy
Annapolis, Maryland

Certification of Adviser(s) Approval

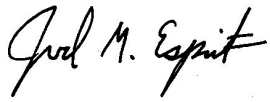
Assistant Professor Matthew G. Feemster
Department of Weapons and Systems Engineering



---

3 May 2007

Assistant Professor Joel M. Esposito
Department of Weapons and Systems Engineering



---

3 May 2007

Acceptance for the Trident Scholar Committee

Professor Joyce E. Shade
Deputy Director of Research & Scholarship

---

3 May 2007

USNA-1531-2

1. Abstract

The goal of this Trident project was to develop an independent control scheme to allow a team of autonomous tugboats to move a large disabled vessel, such as a barge, to a desired position and orientation. Independence refers to the extent to which each tugboat's actions were free from knowledge of the locations and actions of other tugboats. Performance of the team was quantified by measuring the positional error and time required to affect the motion, while respecting maximum power constraints on the thrust. Applications of the project include difficult or dangerous tasks such as moving disabled vessels or vessels "not under command" through hostile or dangerous areas, and transportation of large objects such as marine construction equipment, off-shore bases, drilling platforms, and sonar arrays.

Although it would be ideal to increase both the independence and performance of the system, it must be realized that by increasing one of these, the other is typically degraded. In order to measure performance, a control strategy (the baseline) was designed that required the attachment points of all tugboats to be known. However, this architecture was not desirable, since it was less independent of system knowledge. In contrast, to allow for the elimination of known tugboat location, an adaptive control strategy was developed which resulted in degradation of performance. These two Scenarios were explored and in the course of solving them, the tradeoff between performance and independence was quantified.

To the author's knowledge, this is the first study of its kind and complexity. Although previous work has studied adaptive control of a multi-input and multi-output system, its extent and focus was not close to this research. Each tugboat used on-line adaptive control methods to compensate for the unknown actions of other swarm members. The analysis was verified through simulation. In addition, an experimental proof-of-concept device was built and in-water

experiments were used to validate the results. An incremental approach to experiment design was used to mitigate the challenges of in-water experimentation.

Keywords: Adaptive Control, Swarm Robotics, Parameter Identification, and Automatic Control

2. Acknowledgments

First and foremost, I would like to thank both of my advisors: Professor Feemster and Professor Esposito, both of the U.S. Naval Academy's Systems Engineering Department. Their help and guidance, pertaining to both issues regarding my Trident project and issues regarding other topics, has proven invaluable. I especially am thankful for their advice on my future plans of development, such as graduate school and opportunities after military service. Their unique sense of humor has routinely made otherwise dry topics very entertaining and fun to learn. Next, I would like to thank the Weapons and Systems Engineering Technical Support Department (TSD). Staff members including Joe Bradshaw, Ralph Wicklund, and Norm Tyson have been very helpful in assisting me in the design, building, and implementation of my experimental apparatus. I make several trips per day down to the TSD room for parts and advice, and my experimental apparatus would still be a pile of parts if it were not for their help. Finally, I would like to thank Professor Shade and the members of my Trident subcommittee: Associate Professor Sarah Mouring, CAPT Webster-Giddings, and Associate Professor Craig Whitaker. I truly appreciate the amount of time and consideration each one of you put into reading and analyzing my report and presentation. Your effort is what keeps the Trident Scholar program the premier undergraduate research opportunity at the United States Naval Academy.

3. **Table of Contents**

1. Abstract 1
2. Acknowledgments 2
3. Table of Contents 3
4. List of Figures 4
5. List of Tables 4
6. Introduction 6
   6.1 Problem Formulation 8
   6.2 Related Work 11
   6.3 Problem Approach 12
7. Control Design 13
   7.1 Dynamic Model 14
   7.2 Control Algorithm Development 17
   7.3 Initial Condition Dependency 19
   7.4 Basics of Adaptive Control 20
   7.5 Persistent Excitation 20
8. Controller solving Scenario I (known tugboat locations) 21
   8.1 Controller I Derivation and Proof 21
   8.2 Controller I Simulation Results 26
   8.3 Controller I Experimental Results 28
9. Controller solving Scenario II (unknown hydrodynamic drag) 32
   9.1 Controller II Derivation and Proof 32
   9.2 Controller II Simulation Results 36
   9.3 Controller II Experimental Results 39
10. Controller solving Scenario III (unknown tugboat locations) 42
   10.1 Controller III Derivation and Proof 42
   10.2 Controller III Simulation Results 46
   10.3 Controller III Experimental Results 48
11. Performance Analysis, Independence Analysis, and Controller Comparison 50
   11.1 Performance Metrics 50
      11.1.1 Settling Time 50
      11.1.2 Positional Error 52
      11.1.3 Thrust Conservation 53
   11.2 Performance Comparison 54
   11.3 Independence Analysis 56
   11.4 Controller Comparison 56
12. Simulation 58
   12.1 S-function 58
13. Experimental Vessel Design and Construction 60
   13.1 Vessel Design 60
   13.2 Vessel Internals 63
      13.2.1 Batteries 63
      13.2.2 Control Board 64
   13.3 Vision System 67
   13.4 System Integration 69

13.4.1 Serial Communications 69
13.4.2 Shell Code 71
13.5 Large Scale Experimental Vessel 71
14. Conclusion 73
14.1 Contributions 73
14.1.1 Contributions to Control Systems Engineering 73
14.1.2 Contributions to Ongoing Research 75
14.2 Future Work 76
15. Bibliography 81

4. List of Figures

Figure 1: Swarm of autonomous tugboats (ovals) manipulating a disabled ship to port. 6
Figure 2: SNAME (1950) marine motion variables. 15
Figure 3: Diagram of reference frames. 16
Figure 4: Utilized Swarm Configuration 23
Figure 5: Full Controller Simulation Results for IC - 3 27
Figure 6: Full Controller Experimental Results of IC-3 30
Figure 7: Adaptive Drag Controller Simulation Results for IC-3 37
Figure 8: Adaptive Drag Controller Simulation Parameters for IC-3 39
Figure 9: Adaptive Drag Controller Experimental Results for IC-3 40
Figure 10: Adaptive Drag Controller Experimental Parameters for IC-3 41
Figure 11: Adaptive B Controller Simulation Results for IC-3 47
Figure 12: Adaptive B Controller Simulation Parameters 47
Figure 13: Adaptive B Controller Experimental Results for IC-3 49
Figure 14: Adaptive B Controller Experimental Parameters 49
Figure 15: Base S-function code 59
Figure 16: Original Vessel Schematic 61
Figure 17: Experimental Vessel (Top) 62
Figure 18: Experimental Vessel (Side) 63
Figure 19: “Control Board” of the Vessel 64
Figure 20: Control Board Schematic 65
Figure 21: Vessel LEDs used for Light Invariant Tracking 68
Figure 23: Large Scale Experimental Vessel 72

5. List of Tables

Table 1: Dynamic Model Nomenclature. 15
Table 2: Initial and Final Condition Sets 19
Table 3: Variable Definitions for Known Tugboat Position Example 22
Table 4: Variable definitions for Controller II 32
Table 5: Variables used for the derivation of Controller III. 43
Table 6: Tolerance vector values and corresponding normal values for each set of initial conditions 51
Table 7: Settling Time (seconds) for each Controller. 51

Table 8: Positional Error (meters) for each Controller. 53
Table 9: Thrust Conservation (Newtons) for each Controller. 54
Table 10: Independence metric (number of variables) for each controller 56

6. Introduction

The term "swarm robotics" refers to using a group of small and relatively inexpensive robots to complete complex tasks through communication and coordination rather than through task-specific tooling of a single more sophisticated robot. A swarm's decision making occurs in a decentralized or distributed fashion (i.e., there is no central lead robot) much like a swarm of ants or bees. Each member of the group decides its own actions based on the information received, either through sensors or explicit communication with other members of the swarm. Alone, each of the swarm members is incapable of performing the task successfully, but the whole is far more than the sum of the parts. The advantages of swarm robots are many, including increased robustness and survivability (due to decentralized decision making), lower cost, and increased mission adaptability [1].

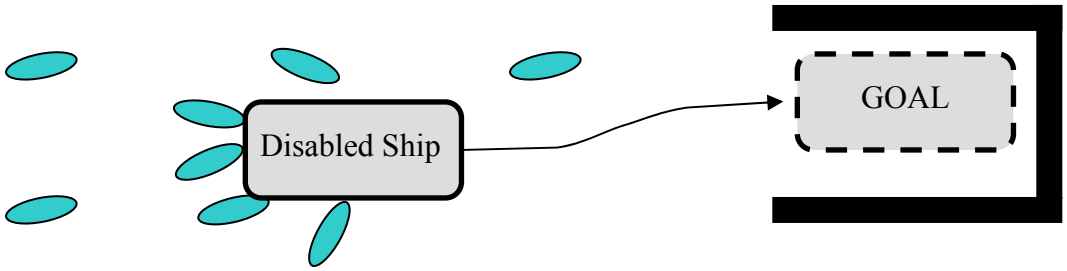


Figure 1: Swarm of autonomous tugboats (ovals) manipulating a disabled ship to port.

The goal of this Trident project was to design a control and coordination strategy to allow a swarm of autonomous tugboats to manipulate a barge or disabled ship as depicted in Figure 1. This application was ideal for swarm robotics since it is impossible for a single boat to complete the task due to such aspects as thrust limitations of each swarm member. Applications of the project include difficult or hazardous tasks such as moving disabled vessels or vessels "not under command" through hostile or dangerous areas, and transportation of large objects such as marine construction equipment, off-shore bases, drilling platforms, and sonar arrays. Knowledge gained

by research from this project will benefit the Navy in the following ways: tugboat manpower reduction, semi-automation of a communication intensive and potentially deadly evolution, and reduced cost. In addition, knowledge gained from this project will give a single operator total control over the team of tugboats, rather than the Navy's current arrangement of the pilot coordinating the actions of each tugboat crew through radio communication. Also, control schema developed during this project will have possible applications in other areas of swarm robotics such as object manipulation with land-based robots, manipulation with space-based robots, and micro-manipulation with small scale robots.

Because swarms do not have a centralized decision maker or lead robot, any information an individual robot needed to know about other members of the swarm in order to complete its task must come through sensor information or wireless communication. However, strong dependence on wireless links decreases independence by making the system more susceptible to interference, jamming, noise, or loss of a swarm member. Truly distributed operation would not require any information passing between the swarm members or preexisting knowledge about the placement of the swarm members, offering the ultimate Scenario with regards to independence. On the other hand, performance of the swarm generally improved with more shared knowledge and coordination, such as each boat knowing the exact position and intention of every other boat. However, this architecture was not desirable, since it is less independent of tugboat position inaccuracies, the failure of a single tugboat or loss of the wireless network. Although it would be ideal to increase both the independence and performance of the system, it must be acknowledged that by increasing one of these, the other is typically degraded. A central challenge in this area was to maximize performance of the team without sacrificing independence. This Trident project specifically dealt with the control aspects inherent in swarm manipulation of a barge.

Success was measured by the time required to move the large vessel from starting point to finish using different levels of data exchange (known information). The project's goal was to systematically "decentralize" the control strategy from perfect knowledge (all-to-all exchange of position and thrust information over the wireless network) to the most extreme level of decentralization that was possible (no messages exchanged between tugboats).

6.1 Problem Formulation

The swarm motion was conducted in two phases.

Phase 1. Each tugboat will establish physical contact with the barge by moving to a desirable point around the barge's hull.

Phase 2. Each tugboat will use a combination of information gathered from sensors or communication with its peers to calculate its thrust magnitude and direction in order to move the barge to its desired position and orientation.

In this project, Phase 1 has already occurred, so the focus will be directed to Phase 2. In Phase 2, It was assumed that, each *tugboat* knows:

- Its own location and orientation with respect to the *barge*,
- Its thrust capabilities (maximum magnitude and direction range),
- The current location of center of mass and orientation of the *barge*,
- The desired location of the center of mass and orientation of the *barge*, and
- The physical properties of the *barge* such as geometry, displacement/weight, drag coefficient, and added mass.

In addition, it was assumed that:

- Each tugboat was securely attached to the barge and no slipping was occurring,
- Each of the tugboats was identical in its minimum/maximum thrust capabilities,

- The mass and drag coefficients of a tugboat was negligible when compared to that of the barge,
- The barge was disabled (not powered), and
- The tugboats were equipped with a method of sending peer-to-peer messages to other tugboats (if needed).

Within this project, we defined *Performance* and *Independence* in the subsequent manner in order to quantify a controller's efficacy. *Performance* was measured by determining the amount of time it took the tugboat swarm to move the barge from its initial position to its final settling position taking into account positional error and thrust conservation. *Independence* was a measure of control strategy decentralization and was measured by determining the extent to which each controller was free from the knowledge of the locations and actions of the other tugboats in order to manipulate the barge. *Independence* was quantified in the controller's derivation by determining the terms that could be estimated by the controller, rather than measured.

In order to explore the trade-off between *performance* and *independence*, control strategies utilizing three levels of given information were investigated. For each Scenario, the goal was to move the barge to a desired position and orientation. Each tugboat had to compute the magnitude and direction of its thrust, knowing all of the quantities listed above in addition to the information below:

- **Scenario I** (known tugboat locations): Each tugboat knew the number of swarm members in contact with the barge, their positions, the barge's hydrodynamic drag, and their thrust directions but did not require knowledge of the other tug's thrust magnitude. This Scenario was considered the *performance* baseline as we expected to obtain the best

results.

- **Scenario II** (unknown hydrodynamic drag): Each tugboat knew the number of swarm members in contact with the barge, their positions, and their thrust directions but did not know hydrodynamic drag and other swarm member's thrust magnitude. The hydrodynamic drag estimates were then updated on-line in real time to properly manipulate the barge.
- **Scenario III** (unknown positions): Each tugboat knew the number of swarm members in contact with the barge and hydrodynamic drag but did not know other swarm member's thrust magnitude, thrust direction, and position. As it turned out, some knowledge of the sign of each location parameter was required. This knowledge was mainly needed to determine how the thrust of each tugboat affected the orientation of the system by determining the direction of the net torque. In essence, each tugboat knew qualitative information about how its actions were going to affect the whole system but did not know the quantitative location of the other tugboats and could adapt its location estimates in real time to properly manipulate the barge.

The three Scenarios presented above differ from the original four Scenarios proposed because in the course of study it was found that thrust magnitude between tugboats was not coupled. Coupling meant that the thrust magnitude of one tugboat depended on the thrust output of another tugboat. In each of the controllers presented below, the thrust calculation of one tugboat did not depend upon the thrust output of another tugboat but was purely a function of the tugboat location, dynamic parameters, and positional error. The original four problem formulation Scenarios are given below:

- **Original Scenario I** (all-to-all exchange): Each tugboat knows the number of swarm

members in contact with the barge, their positions, and their thrust magnitude and direction (most centralized). This Scenario is similar to a vessel equipped with fixed thruster pods and therefore can be considered as the *performance* baseline.

- **Original Scenario II** (known positions and orientations): Each tugboat knows the number of swarm members in contact with the tugboat, their positions, and their thrust direction and does not know other swarm member's thrust magnitude.
- **Original Scenario III** (known positions): Each tugboat knows the number of swarm members in contact with the barge and their positions along the hull and does not know the other swarm member's thrust direction and magnitude.
- **Original Scenario IV** (truly decentralized): Each tugboat only knows the number of swarm members in contact with the barge.

As explained above, the de-coupled system allowed the four original Scenarios to be simplified into three new Scenarios. Also, it was found that Original Scenario IV could not be solved in a closed form expression. Some *a priori* knowledge of tugboat locations was needed to properly control the barge/tugboat system. This knowledge was needed to make sure the tugboats were pushing in the correct direction to properly influence the rotation of the barge/tugboat system.

6.2 Related Work

Swarm control is a very active area of research [2, 3, 4]. Similarly, other Trident Scholars have examined swarm control problems. In Bishop's work, various behavior-based control techniques are combined to create a novel controller designed to search for mines [5]. In Esposito's work, the problem of maintaining connectivity of a wireless network for a swarm of land based robots was addressed [6]. However, all works focus on position control of the swarm

to perform tasks such as searching, reconnaissance, and traveling to a goal position. In this Trident project, the swarm will operate on the dynamic level where forces and torques will be generated in order to move an object.

A second area of active research is "robot pushing," first analyzed for a single robot [7]. Dynamically manipulating objects using two or three robots was examined in two previous works [8, 9]. In both cases, it is unclear how to extend the methodologies to many robots with decentralized decision making. A different approach to this problem is explored primarily using caging algorithms [10, 11, 12]. Controllers are designed which force robots to surround the object. Inter-robot spacing is constrained to be small enough that it is impossible for it to "escape", meaning that as the robots move, so must the object. While this approach is decentralized, the primary drawback is that it is strictly for land based robots. The problem is treated as a position control problem, ignoring dynamic forces. The extension to water manipulation requires consideration of hydro-dynamic forces, drift, and disturbances.

As mentioned, Scenario I will be considered a baseline for comparison by the other Scenarios. In fact, a situation close to Scenario I was solved both theoretically and experimentally [13]. Fossen's work investigates a situation close to Scenario I with an experiment using an apparatus similar to the one constructed to reduce power consumption and increase maneuverability through singularity avoidance [14]. Essentially, this technique maximizes lever arms to reduce the required amount of power input by the thrusters [14]. Scenarios I and II have been further explored by Webster in the work entitled, "Optimum allocation for multiple thrusters" [15].

6.3 Problem Approach

The three major parts of the project were:

1. Control design,
2. Simulation, and
3. Experimentation.

The author initially focused efforts on the construction of the experimental vessel due to its inherent tendency to take more time to fabricate than previously estimated. However, while focusing on experimentation, the theoretical counterpart of this project was not neglected, which included both control design and simulation. In the spring semester, the author devoted the majority of time to developing control strategies targeted at addressing the issues of Scenarios I through III. After developing these controllers, they were subsequently simulated in Matlab to prove viability, and further, developed their practical application by coding the controllers onto the base station using Matlab and sending proper thrust commands to a slave C program running on the rabbit microcontroller controlling the experimental apparatus.

This report will detail the experimental vessel construction and discuss the *performance* verses *independence* tradeoff of each controller and make recommendations pertaining to field-ability and reliability of each controller.

7. **Control Design**

The overall control design consisted of three major steps: selecting a representative three degree of freedom model (x, y for translation and ψ for rotation), developing and proving control algorithms, and quantifying the tradeoff between *performance* and *independence*. The author first selected a suitable model and has developed infrastructure code in which to simulate and implement the designed control algorithms. The author also learned how to derive, prove, and simulate adaptive update laws and controllers. Controllers that solved Scenarios I through

III were also developed, simulated, and proven. The Scenario I baseline controller was used as a performance comparison for the more *independent* controllers of Scenarios II and III.

7.1 Dynamic Model

Suitable coordinate frames and a dynamic model were found in Fossen's book entitled, "Marine Control Systems" [16]. The following describes two potential reference frames. [16]

- ECEF (e-frame) The Earth-centered Earth-fixed (ECEF) reference frame was affixed to the center of the earth, but it rotated along with the earth. This frame was analogous to the camera/fixed frame described later in this report. The camera frame's origin was affixed where the camera was mounted. The camera frame has all of the properties of the ECEF frame except that it was not affixed to the earth's center. [16]
- BODY (b-frame) this was the frame in which linear and angular velocities were defined. Position and orientation were described relative to an inertia reference frame such as the e-frame. The origin of the b-frame was affixed to the center of gravity of the barge. Each axis was defined the following way (see Figure 2): x_b points from aft to fore, y_b points to starboard, and z_b points from top to bottom. This reference frame was used in the project. [16]

Each of these reference frames is displayed in Figure 3. To convert between reference frames, homogeneous transformation matrices were utilized. [16]

According to the established convention used for marine vessels named SNAME (1950), the following nomenclature applied.

Degrees of Freedom		forces and moments	linear and angular velocities	positions and Euler angles
1	x-direction motions (surge)	X	*u*	*x*
2	y-direction motions (sway)	Y	*v*	*y*
3	z-direction motions (heave)	Z	*w*	*z*
4	rotations about x-axis (roll,heel)	K	*p*	*Φ*
5	rotations about y-axis (pitch,trim)	M	*q*	*θ*
6	rotations about z-axis (yaw)	N	*r*	ψ

Table 1: Dynamic Model Nomenclature.

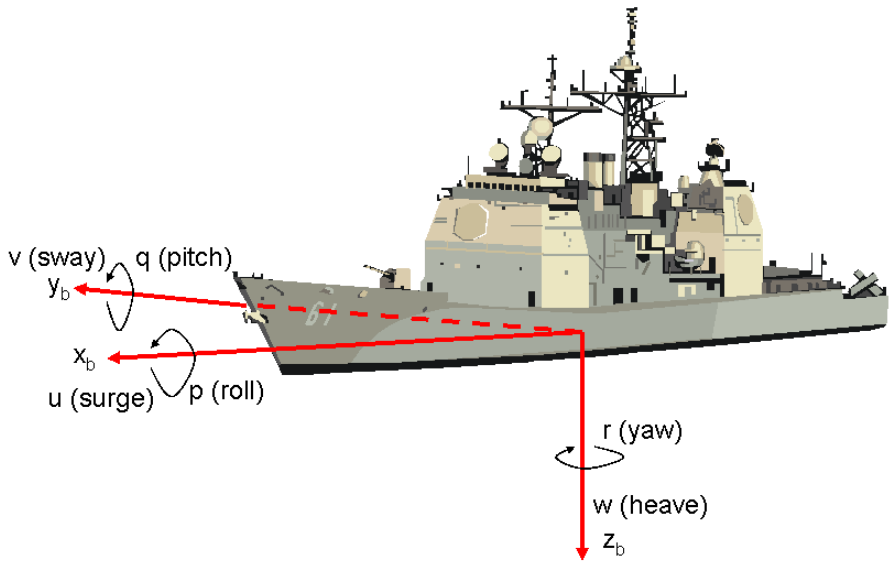


Figure 2: SNAME (1950) marine motion variables.

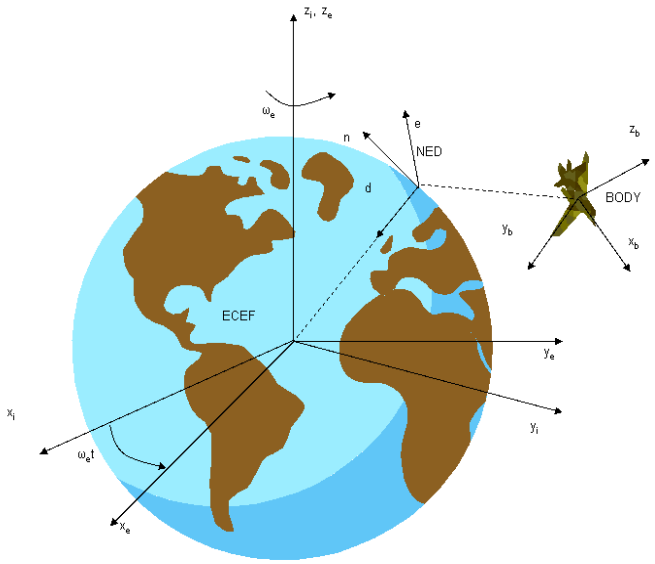


Figure 3: Diagram of reference frames.

Figure 3 shows the location and orientation of these variables relative to a ship schematic. The marine motion variables in Figure 3 are defined with respect to the body-fixed reference frame.

A simplified model described by [16] which was suited for this project was the three degree of freedom (3-DOF) model for surface vessels. This model neglected heave, roll, and pitch based on the assumption that these variables were small. This assumption was suitable for most ships in harbor conditions and was adequate for the design purposes. The following equations describe the 3-DOF model found in [16],

$$\dot{P} = R(\psi)v$$
$$M\dot{v} + Dv + F_d = Bu.$$

Where:

- $R(\psi) = R_{z,\psi} = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}$ represented a rotation matrix whose purpose was to relate b-frame quantities to n-frame quantities.
- $v = \begin{bmatrix} u & v & r \end{bmatrix}^T$ represented the linear and angular velocities measured with respect to the b-frame.
- $P = \begin{bmatrix} x & y & \psi \end{bmatrix}^T$ denoted the positions measured with respect to the n-frame.
- The M matrix [3 x 3] described the mass of the barge (including the effects of added mass), which was a quantity measured experimentally with aid from the Hydromechanics Laboratory.
- The D matrix [3 x 3] represented the effects of damping in the surge direction and was decoupled from sway and yaw motion. [14]
- The vector F_d [3 x 1] captured any disturbances (*i.e.*, waves, wind, *etc.*). In order to simplify the model, $F_d = 0$ for this investigation.
- u was the thrust input vector [N/2 x 1] from the swarm members (where N denotes the number of tugboat opposing pairs). This matrix gives the magnitude of each swarm pair's thrust.
- The B matrix [3 x N/2] described thruster configuration which was dependent on the contact location and orientation of the other tugboats and was considered partially unknown in Scenario III. This will be described in more detail later.

7.2 Control Algorithm Development

In this portion of the project, a suitable control strategy for each swarm member was

designed subject to the knowledge constraints outlined in Scenarios I though III of the Problem Formulation. The development of these control strategies represented the main thrust of the research in this project during the spring semester. With the level of information available to each swarm member decreasing through the various Scenarios, the effort of the control design was focused on how to handle this reduction of information as the swarm architecture approached a decentralized structure while still positioning and orienting the disabled barge. To be able to accomplish this task, the technique of adaptive control was explored since this strategy lends itself to being able to compensate for *constant but unknown* system parameters values [13]. An issue encountered in Scenario II and Scenario III was the problem of unknown parameter values. In Scenario II, the hydrodynamic drag of the barge was unknown. The control algorithm developed used on-line adaptive update laws to estimate drag and move the barge to the desired endpoint. Under the constraints of Scenario III, each swarm member was only aware of the number of members in the swarm and the hydrodynamic drag. Each member may not know where in relation to the disabled ship's center of mass the swarm vehicle had attached. This lack of knowledge of the vehicle's attachment point was captured by the above dynamic model in the sense that elements in B were unknown. If it was assumed that no slipping occurred, then these unknown parameters were *constant*. Therefore, the area of *adaptive control* [13] afforded techniques to compensate for unknown, constant parameter value. For this case, the adaptive controller monitored the translation and orientation tracking errors and then made on-line adjustments to minimize these error signals.

To investigate the developments of control algorithms for Scenarios I, II, and III, the basics of adaptive control had to be researched. In the following chapters, the details of the derivation and proof of several adaptive controllers will be presented; including a three degree-

of-freedom controller, a three degree-of-freedom adaptive update law to compensate for unknown hydrodynamic drag, and a three degree-of-freedom adaptive update law to compensate for unknown tugboat placement.

7.3 Initial Condition Dependency

One important observation to note was the controllers' dependency on the initial conditions of the vessel before the control algorithm was initiated. The time to the desired endpoint, ability of the control to take a direct path to the endpoint, and the steady-state error all depended heavily on the initial state of the vessel in relation to its desired endpoint. For the simulation and experimentation parts of this project, the author chose three sets of initial conditions and endpoints in which to test all three controllers. These sets of criterion were then simulated and experimented with all three controllers. For the sake of brevity, the author will only present the third set of initial conditions because it gave the best overall representation of both displacement and orientation control. The other sets of initial conditions, one and two, will be detailed in the performance analysis and conclusion sections. The values for all the initial condition sets are given in Table 2.

Condition Set	Initial 1	Final 1	Initial 2	Final 2	Initial 3	Final 3
X position (m)	3.58	4.4	2.16	2.0	5.58	2.0
Y position (m)	0.80	2.2	2.96	2.0	0.30	2.0
Heading (degrees)	210	100	321	270	83.6	90

Table 2: Initial and Final Condition Sets

7.4 Basics of Adaptive Control

As explained above, adaptive control is a technique used to account for an unknown but constant parameter. Adaptive control uses the error signals defined in a system to constantly change the parameter estimate in a way in which it forces the error signals to zero. This is done by defining the parameter estimates' update law in terms of the system error. As the error changes the parameter estimate, the parameter estimate changes the output of the controller. This output then affects the error of the system, and the whole iterative process begins again. Essentially, an adaptive controller varies the parameter estimate to see how it will affect the error of the system. It then updates, or adapts, its parameter estimate based on how the system reacts to the previous parameter estimate. It is important to realize that an adaptive controller does not necessarily find the actual value of the parameter. This is explained below in Section 7.5.

7.5 Persistent Excitation

In some cases, the adaptive update laws eventually drive the system parameter estimates to their actual value. It is important to note that this does not always happen when using adaptive control. The goal of adaptive control is to drive the system's output, be it velocity, acceleration, or position, to a desired output. The goal of adaptive control is not to determine the actual system parameter that was being changed in the adaptive update law. Adaptive controllers use whatever system parameter estimate that is needed for the system to track its desired output. The below derivations never determined the actual parameter values due to a condition called persistent excitation [13]. Persistent excitation means that the input of the system, the desired output, must have been constantly moving for the adaptive update laws to determine the actual system parameters [13]. In the below examples, all of the inputs (desired outputs) are constant velocity. A constant velocity does not meet the condition of persistent excitation; therefore, the

examples do not find the actual parameter values. The condition of persistent excitation is not important for the project, because its goal is not to determine the actual system parameters; it is to follow a desired track. Persistent excitation does not affect the vessel's ability to track a desired output.

8. Controller solving Scenario I (known tugboat locations)

After a necessary explanation of the background work pertaining to control design, simulation, and experimental vessel design and construction, the author will now present the control algorithm derivations, proofs, simulation results and experimental results. Each controller will be explained and documented in a separate subsequent chapter, and then *performance* vs. *independence* will be analyzed in the following chapter for each controller. Each following chapter will solve Scenario I through Scenario III which were detailed earlier.

8.1 Controller I Derivation and Proof

To properly actuate the vessel, the tugboat placement configuration of Figure 4 was selected and used in the derivation, simulation, and experimentation. This placement allowed full controllability with the particular constraints on the system that were defined in 6.1. Particularly, the configuration of Figure 4 represents only one possible selection from a large set of possible configurations that could control the barge. This swarm configuration is then used to define the thrust input and configuration matrices given in equation (1.1). The commutation strategy of equation (1.2) is needed to account for the fact the tug boats can only exert a positive thrust vector.

Variable	Definition	Variable	Definition	Variable	Definition	Variable	Definition
M [3x3]	Mass of the vessel, Includes added mass and moment of inertia	e [3x1]	Positional error	ψ [1x1]	Yaw angle (measures orientation)	α [3x3]	Position gain
$\dot{v}$ [3x1]	Vessel acceleration	$\dot{P}_d$ [3x1]	Desired velocity	$\dot{P}$ [3x1]	Global velocity	$V(t)$ [1x1]	Lyapunov function
D [3x3]	Hydrodynamic drag	$\dot{e}$ [3x1]	Velocity error	k [3x3]	Gain term	$\dot{V}(t)$ [1x1]	Lyapunov function derivative
v [3x1]	Vessel velocity	P_d [3x1]	Desired Position	P [3x1]	Global position	I [3x3]	Identity matrix
r [3x1]	Filtered tracking error	R [3x3]	Rotation matrix (converts from global to body frame)	B_s [3x3]	Defined swarm thruster configuration	R^T [3x3]	Transpose of the rotation matrix
U_s [3x1]	Swarm thrust magnitude input	K_r [3x3]	Error Gain				

Table 3: Variable Definitions for Known Tugboat Position Example

Vessel configuration boundary conditions:

$$r_4 = r_1 \qquad \alpha_1 = 0° \qquad \theta_1 = 0°$$
$$r_5 = r_2 \qquad \alpha_4 = 180° \qquad \theta_4 = 180°$$
$$r_6 = r_3 \qquad \alpha_2 = 270.0° \qquad \theta_3 = 135.0°$$
$$\alpha_5 = 90.0° \qquad \theta_6 = 360° - \theta_3$$
$$\alpha_3 = 270.0° \qquad \theta_2 = 90°$$
$$\alpha_6 = 90.0° \qquad \theta_5 = 270.0°$$

(1.1)
$$B_s U_s = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & -1 \\ 0 & -r_2\cos(\theta_2) & 0 \end{bmatrix} \begin{bmatrix} u_1 - u_4 \\ u_2 - u_5 \\ u_3 - u_6 \end{bmatrix}$$

(1.2)
$$u_i = \frac{1}{2}(U_s(i,1) + \sqrt{U_s(i,1)^2 + \gamma_0{}^2}$$
$$u_{i+3} = \frac{1}{2}(-U_s(i,1) + \sqrt{U_s(i,1)^2 + \gamma_0{}^2}$$

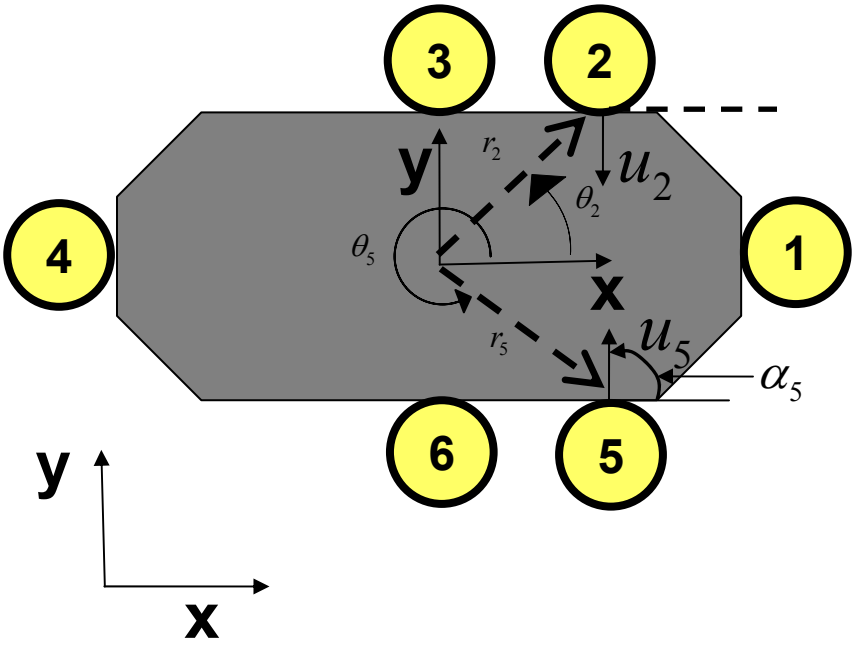


Figure 4: Utilized Swarm Configuration

This section will provide the derivation of a control algorithm that successfully manipulated the system configuration in Figure 4 of known tugboat locations, number of swarm members, thrust directions, and hydrodynamic drag. The goal of the derived controller was to drive the filtered tracking error between the desired system position and actual system position of the barge to zero. When the error reached zero, this meant that the system was behaving as desired and the control strategy was effective. First, the author defined each variable that was used in the following known tugboat location controller and this is shown in Table 3. To start the derivation, the aforementioned system model was manipulated to include an error term consisting of the difference between the user defined desired inertial position P_d and velocity $\dot{P}_d$ and the actual system position P and velocity $\dot{P}$. The system error equations are

$$\begin{aligned} e &= P_d - P \\ \dot{e} &= \dot{P}_d - \dot{P} \\ r(t) &= \dot{e} - \alpha e. \end{aligned} \tag{1.3}$$

The dynamic system model that is given in (1.4) uses the transformation matrix given in (1.5) to convert from the body frame to the global frame. This is done by using the derivative of (1.5), and the result is (1.6). After solving the transformation matrix for v, the result is plugged into (1.6) and (1.7) is obtained after simplification and insertion of the dynamic model for $\dot{v}$. Equation (1.7) is essentially the global acceleration of the system which takes into account barge dynamics.

$$M\dot{v} = Dv = B_s U_s \tag{1.4}$$

$$\dot{P} = Rv \tag{1.5}$$

$$\ddot{P} = \dot{R}v + R\dot{v} \tag{1.6}$$

$$\ddot{P} = -\psi \times \dot{P} + RM^{-1}(-Dv + B_s U_s) \tag{1.7}$$

After taking the double derivative of the filtered tracking error, r, equation (1.8) is the open-loop filtered tracking error dynamics. This process is then continued by substituting equation (1.7) into the error equation of (1.8) to produce a rough form of the open-loop filtered tracking error dynamics given in (1.9). To further transform the open-loop filtered tracking error dynamics to the global frame, equation (1.5) is used. This previous step insures that the velocity used in the equation is the global velocity obtained by GPS, not the body velocity obtained by an inertial measurement unit. Body velocity could be used directly in the equation; however, only global velocity was available for the system.

$$\dot{r} = \ddot{P}_d - \ddot{P} + \alpha\dot{e} \tag{1.8}$$

$$\dot{r} = \ddot{P}_d + \psi \times \dot{P} + RM^{-1}DV - RM^{-1}B_S U_S + \alpha\dot{e} \tag{1.9}$$

$$\dot{r} = \ddot{P}_d + \alpha\dot{e} + \dot{\psi} \times \dot{P} + RM^{-1}DR^T\dot{P} - RM^{-1}B_S U_S \tag{1.10}$$

The overarching goal is to drive the filtered tracking error, the errors of both the position and velocity, to zero. If the filtered tracking error is zero, then it can be shown that $e(t)$ and $\dot{e}(t)$ are also zero which means that the vessel has arrived at the desired location. In all of the experimental trials this meant that once the filtered tracking error was zero, the barge was at the desired point with zero remaining velocity. In order to drive the filtered tracking error to zero, the system needs the following:

1. P and $\dot{P}$ are readily available for measurement,
2. M is positive definite and symmetric, and
3. R exhibits the following properties: $R^T R = I_3$, $\dot{R} = -\dot{\psi} \times R$, and $\|R\| = 1, \forall \psi$.

Each of the previous criteria ensures that the controller avoids singularities and is solvable. If any of the criteria is not valid, controller implementation is not possible since the inverse of M is required. If the required matrices can not be inverted, then the entire controller will exhibit a singularity. For almost all mechanical systems, M will have full rank and be positive definite.

To obtain the proper mathematical expression for the system input, the author solved equation (1.10) for the thrust input vector, U_s. The result is now called Controller I and is given in (1.11). To check the stability of the system and make sure that Controller I always drives the filtered tracking error to zero, equation (1.11) is substituted back into the open-loop filtered tracking error dynamics given in equation (1.10) and the expression in (1.12) is obtained.

$$U_S = \left[RM^{-1}B_S \right]^{-1} \left[\ddot{P}_d + \alpha \dot{e} + K_r r + \dot{\psi} \times \dot{P} + RM^{-1}DR^T \dot{P} \right] \tag{1.11}$$

$$\dot{r} = -K_r r \tag{1.12}$$

It is important to note that equation (1.12) follows the criteria for a globally exponentially state equation because its solution is $r(t) = e^{-K_r t}$ if $K_r \geq 0$. This criterion ensures that any presence of the filtered tracking error will force its value back to zero in an exponential fashion. The

definition of the filtered tracking error given in equation (1.8) ensures that as the filtered tracking error is driven to zero, the positional error and velocity error also approach zero. Essentially, this means that if the barge in not where it is supposed to be at the desired speed, it will be forced to the desired location and velocity by the controller of equation (1.11).

8.2 Controller I Simulation Results

The controller that was derived in 8.1 was tested using the S-function simulation code presented in Section 12.1. The only changes made to the S-function shell code occurred in the previously detailed code sections. These changes entailed entering equations (1.1), (1.2), (1.3), and (1.11) into the existing S-function dynamics as shown in Enclosure 15.1.1. The author then entered the specific control gains, initial conditions, and desired positions associated with Initial Conditions Set Three. Figure 5 illustrates the motion of the disabled barge utilizing the controller of equation (1.11). This figure shows the path and orientation taken by the simulated vessels using the value set of initial conditions three. As shown below, the vessel corrects its heading first and then moves to the desired position while maintaining the desired heading. This plot shows the vessel's holonomic movement capabilities. Holonomic vehicles such as hovercraft and differentially driven vehicles have the capability to turn on a point, meaning they have a turning radius of zero. Steered vehicles are non-holonomic by nature, meaning that they have some finite turning radius and the output of the system can be path dependent.

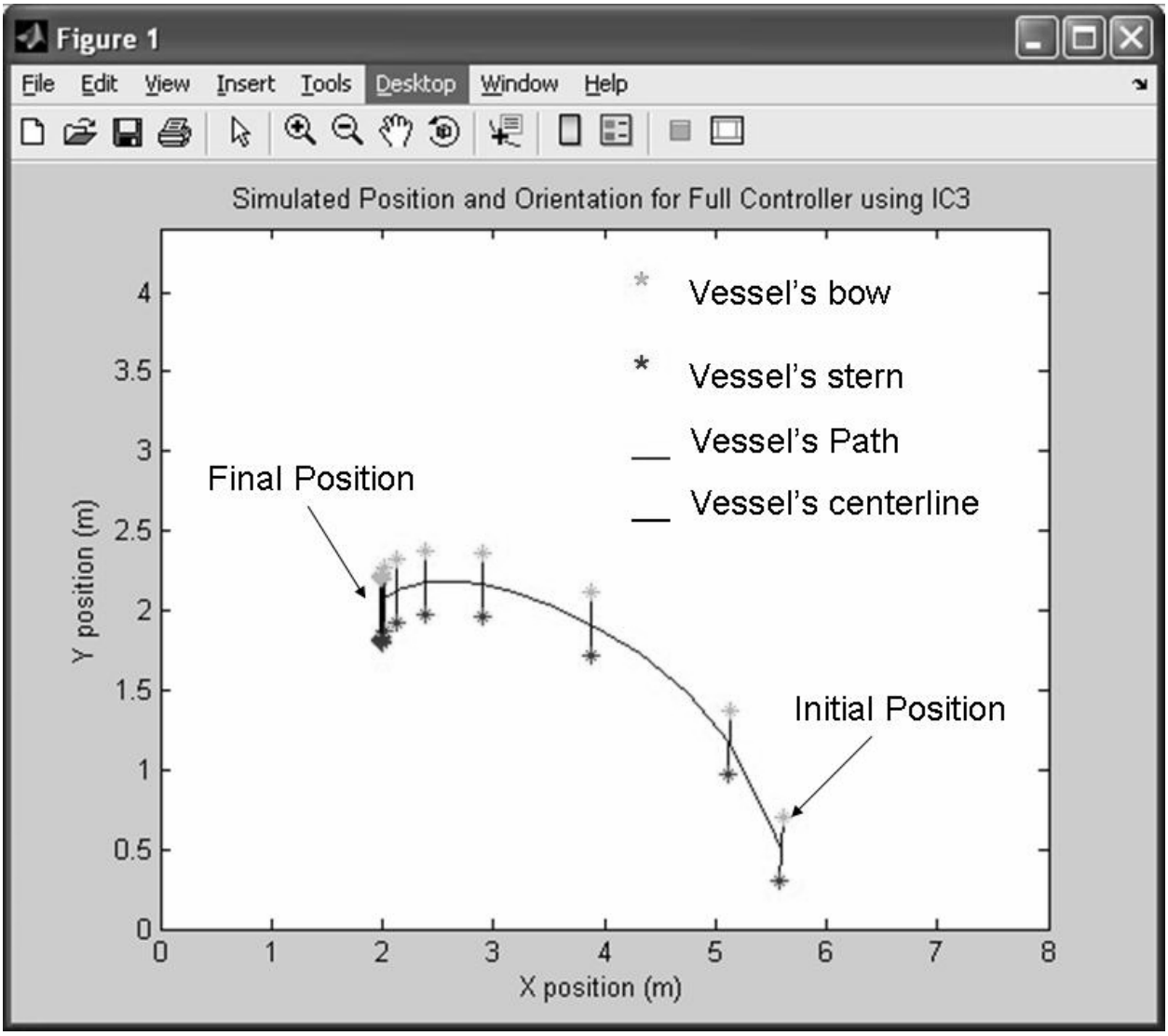


Figure 5: Full Controller Simulation Results for IC - 3

Although the vessel is holonomic, it does have a preferred direction in which movement is easiest. As with most marine vessels, the small-scale experimental vessel's preferred direction of movement is in the direction of the bow. This preferred direction was not taken into account in the control algorithm. The controller is performing regulation, not path tracking; therefore, we can not influence the path the vessel takes to the endpoint. However, we can influence the path if we use a spline trajectory. A spline trajectory is essentially fitting a third-order polynomial curve to a set of locations, velocities, and times. The generated spline trajectory used in this

fitted third-order polynomial curves for both the x and y directions, and then the arctangent of the x and y velocities was used to determine the desired orientation angle. This approach ensured that the bow of the experimental vessel was always pointed in the desired direction of motion. The author decided not to use this approach because it required a predetermined time to the endpoint. Seeing as the performance measure is based on positional steady state error and time to the endpoint, the author decided that generating a spline trajectory would be counterproductive for the objective of this project, determining the tradeoff between system performance and independence.

8.3 Controller I Experimental Results

The controller for known tugboat positions (Controller I) was implemented on the small scale experimental vessel explained in Section 13.1. This controller was inserted into the shell code explained in Section 13.4.2 to obtain the control package in enclosure 15.1.2. The shell code had to be modified to include a timer, integrator, and differentiator. To measure accurate time, the predefined Matlab function tic and toc were used. These functions are essentially a stop watch, tic starts the watch and toc measures the elapsed time from the last tic. To integrate, the trapezoidal rule was used and this equation is given in (1.13).

$$G(x) = (t_2 - t_1)\frac{(f_2(x) - f_1(x))}{2} \tag{1.13}$$

To differentiate, a backwards difference was taken and this equation is given below.

$$\dot{f}(x) = \frac{f_2(x) - f_1(x)}{t_2 - t_1} \tag{1.14}$$

Although the methods for integration and differentiation explained above are straightforward and generally accurate, their use in the control package was not ideal. Both

methods are very susceptible to inaccurate time readings, inaccurate position measurements, and slow control frequencies. If the control gains were not tuned correctly, spikes in the velocity reading would cause the system to go unstable. This generally happened at the end of an experimental run, after the vessel had converged to the desired endpoint. The vessel would be in the process of keeping station on the desired endpoint, and then as soon as an inaccurate velocity measurement was fed into the control algorithm the system would move off the desired endpoint in an erratic fashion and out of the field of view of the camera. This happened for all three controllers because all depend on velocity measurements for control. Future work on the project will include elimination of the velocity measurements through the use of an observer. An observer is essentially a predictive filter. The observer uses measurable quantities from the system to construct a measurement for the unknown state such as translational or rotational velocity. In this project, the observer would use the position of the vessel, along with the dynamic model of the vessel, to construct a measurement for velocity. The author and his advisors currently have an abstract submitted to eliminate velocity measurements in Controller I.

Figure 6 shows the experimental path taken by the vessel. It is important to notice the differences and similarities between the simulated, given in Figure 5, and experimental paths. The vessel's simulated and experimental paths are very similar except for a few differences that include the smoothness of the path, order in which each movement is performed, and time to the steady state position. The simulated vessel path is very smooth and always approaches the desired endpoint while the experimental path has a few fluctuations and seems to overshoot the desired endpoint in the y direction until the almost ninety degree turn. The reason for these disparities could be the mass and moment of inertia matrix used during simulation and experimentation. The same values were used in both simulation and experimentation; however,

these values were very rough calculations and did not include any coupling between terms or any non-linear terms. These disparities were expected during the proposal process, and as stated before, the purpose of this project was not to accurately measure the hydrodynamic properties of the vessel; the purpose was to determine the tradeoffs between performance and independence. These inaccuracies in the model will not affect the performance analysis, because they are prevalent in all three controllers. It is expected that with the same initial conditions and same conditions in the environment, the model inaccuracies remained constant throughout all of the experimental runs.

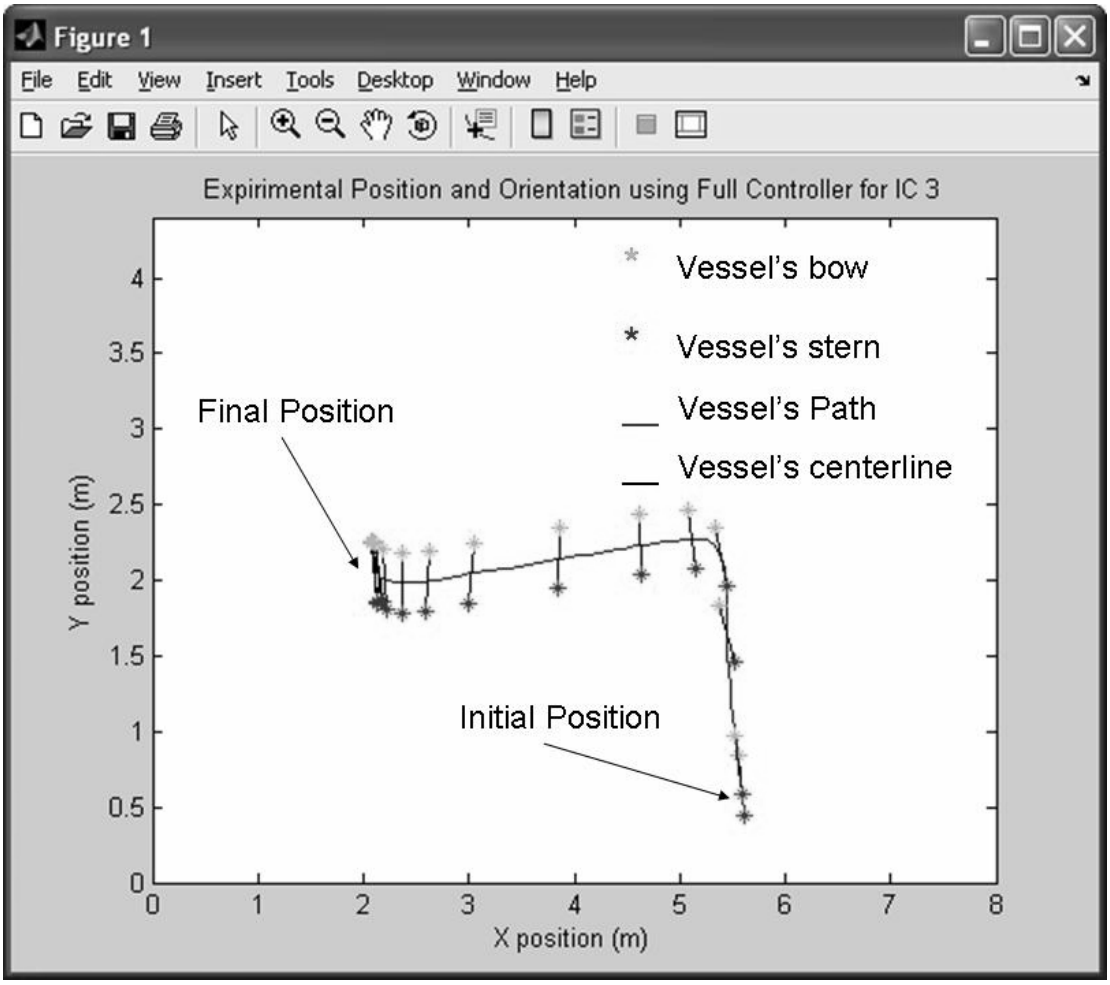


Figure 6: Full Controller Experimental Results of IC-3

The differences between the paths taken in the simulated and experimental runs are also though to be due to the inaccurate mass and drag matrices. The simulated path shows that the controller equally weighs each of the three objectives which include:

1. converging to the desired point in the x direction,
2. converging to the desired point in the y direction, and
3. converging to the desired orientation.

This equal weight on the above objectives yields a smooth path that is always converging to the desired endpoint. The actual path taken is not as smooth as the simulation. Figure 6 shows that the rates of convergence for all three objectives are not the same. The first object reached is converging to the desired orientation. Once in the proper orientation, the controller maintains the orientation while converging to the desired y direction. After reaching the proper y direction and orientation, the controller maintained the previous objectives while converging to the desired point in the x direction. This discrepancy in convergence rates show that the vessel has a preferred direction of motion as stated in Section 8.2. This also intuitively makes sense due to the author's controller gain selection. The author chose the largest gain for the term influencing orientation, the second largest gain for the term influencing y position, and the smallest gain for the term influencing x position. The gains chosen are given below in (1.15).

$$K_r = \begin{bmatrix} 0.2 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & 0.6 \end{bmatrix} \tag{1.15}$$

The inaccuracies in the model as compared to the actual system meant that the simulated vessel reached the desired endpoint in less time than the actual vessel. This was expected because there were many non-linear forces and hydrodynamic drag effects that were not modeled opposing the motion of the vessel. Also, the inaccurate mass matrix meant that the vessel in

simulation could turn easier than the actual vessel. This is shown in the difference of path between the simulated vessel and actual vessel.

9. Controller solving Scenario II (unknown hydrodynamic drag)

The variables used for the following controller derivation and proof are given in Table 4. Controller II solves Scenario II as explain in Section 6.1. Controller II uses adaptive control to estimate unknown drag parameters. After the controller derivation and proof, simulation and experimental results will be given and explained.

9.1 Controller II Derivation and Proof

Variable	Definition	Variable	Definition	Variable	Definition	Variable	Definition
M [3x3]	Mass of the vessel, Includes added mass and moment of inertia	e [3x1]	Positional error	ψ [1x1]	Yaw angle (measures orientation)	α [3x3]	Position gain
$\dot{v}$ [3x1	Vessel acceleration	$\dot{P}_d$ [3x1	Desired velocity	$\dot{\psi}$ [1x1]	Yaw rate	$v(t)$ [1x1]	Lyapunov function
Γ [3x3]	Adaptive update gain matrix	$\ddot{P}_d$ [3x1]	Desired acceleration	r [3x1]	Filtered tracking error	$\dot{r}$ [3x1]	Filtered tracking error derivative
D [3x3]	Hydrodynamic drag	$\dot{e}$ [3x1]	Velocity error	$\dot{P}$ [3x1]	Global velocity	$\dot{V}(t)$ [1x1]	Lyapunov function derivative
v [3x1]	Vessel velocity	P_d [3x1]	Desired Position	k [1x1]	Gain term	I [3x3]	Identity matrix
B_s [3x3]	Vessel thruster configuration	R [3x3]	Rotation matrix (converts from global to body frame)	P [3x1]	Global position	R^T [3x3]	Transpose of the rotation matrix
U_s [3x1]	Swarm thrust magnitude input	K_r [3x3]	Error Gain	B_s [3x3]	Defined swarm thruster configuration	$y(\dot{P})$ [3x3]	Parameter regression matrix
x [1x1]	X position	y [1x1]	Y position	$\dot{\hat{\theta}}$ [3x3]	Parameter estimate vector derivative	θ [3x3]	Parameter vector
$\dot{x}$ [1x1]	X velocity	$\dot{y}$ [1x1]	Y velocity	$\hat{D}$ [1x1]	Drag parameter estimate	$\hat{\theta}$ [3x1]	Parameter estimate vector
$\tilde{\theta}$ [3x1]	Parameter error vector	γ [3x3]	Adaptive update gain	$\dot{\tilde{\theta}}$ [3x1]	Parameter error vector derivative		

Table 4: Variable definitions for Controller II

To derive a proper controller for Scenario II, the system open-loop filtered error dynamics determined in equation (1.10) must be used. This equation is restated in (2.1) using the variables given in Table 4. To manipulate the system so that the drag estimates can be extracted and compensated, the parameterization given in equation (2.2) must be used. In the parameterization process, the $\hat{D}R^T\dot{P}$ matrices are multiplied out to vector form. This vector form is then manipulated in such a way that all drag estimate terms can be pulled out into a 3x1 vector and the remaining terms can be grouped in a 3x3 matrix as shown in equation (2.3). The result of this process is the $y\left(\dot{P}\right)\theta$ term. This term is equal to the original representation but is in the form needed to apply adaptive control.

(2.1) $$\dot{r} = \ddot{P}_d + \alpha\dot{e} + \dot{\psi}\times\dot{P} + RM^{-1}DR^T\dot{P} - RM^{-1}BU$$

(2.2) $$DR^T\dot{P} = \begin{bmatrix} D_1 & 0 & 0 \\ 0 & D_2 & 0 \\ 0 & 0 & D_3 \end{bmatrix}\begin{bmatrix} \cos(\psi) & \sin(\psi) & 0 \\ -\sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} D_1\cos(\psi)\dot{x} + D_1\sin(\psi)\dot{y} \\ -D_2\cos(\psi)\dot{x} + D_2\sin(\psi)\dot{y} \\ D_3\dot{\psi} \end{bmatrix}$$

(2.3) $$\begin{bmatrix} D_1\cos(\psi)\dot{x} + D_1\sin(\psi)\dot{y} \\ -D_2\cos(\psi)\dot{x} + D_2\sin(\psi)\dot{y} \\ D_3\dot{\psi} \end{bmatrix} = \begin{bmatrix} \cos(\psi)\dot{x} + \sin(\psi)\dot{y} & 0 & 0 \\ 0 & \cos(\psi)\dot{y} - \sin(\psi)\dot{x} & 0 \\ 0 & 0 & \dot{\psi} \end{bmatrix}\begin{bmatrix} D_1 \\ D_2 \\ D_3 \end{bmatrix} = y\left(\dot{P}\right)\theta$$

(2.4) $$\dot{r} = \ddot{P}_d + \alpha\dot{e} + \dot{\psi}\times\dot{P} + RM^{-1}y\left(\dot{P}\right)\theta - RM^{-1}BU$$

After the new open-loop filtered tracking error dynamics term with the drag parameter vector was obtained, it was used to solve for the thrust matrix U_S to find the controller given in equation (2.5). This controller was then substituted back into the open-loop filtered tracking error dynamics to obtain equation (2.6). After cancellations, equation (2.6) resolves to the equation given in (2.7), where $\tilde{\theta}$ equals (2.8).

(2.5) $$U_S = \left[R(\psi)M^{-1}B_S\right]^{-1}\left[\ddot{P}_d + \alpha\dot{e} + K_r r + \dot{\psi}\times\dot{P} + R(\psi)M^{-1}y\left(\dot{P}\right)\hat{\theta}\right]$$

$$\dot{r} = \ddot{P}_d + \alpha\dot{e} + \dot{\psi} \times \dot{P} + RM^{-1}DR^T\dot{P} - RM^{-1}B\left[R(\psi)M^{-1}B_S\right]^{-1} \cdot\left[\ddot{P}_d + \alpha\dot{e} + K_r r + \dot{\psi} \times \dot{P} + R(\psi)M^{-1}y\left(\dot{P}\right)\hat{\theta}\right] \tag{2.6}$$

$$\dot{r} = RM^{-1}y\left(\dot{P}\right)\tilde{\theta} - K_r r \tag{2.7}$$

$$\tilde{\theta} = \theta - \hat{\theta} \tag{2.8}$$

After the controller derivation given above, it is now necessary to determine an adaptive update law to find a suitable value of $\hat{\theta}$ for which the controller will be stable. To determine this value, the author will use a Lyapunov function [13]. For a function to fit this definition, it must adhere to the following criteria:

1. The scalar Lyapunov function, $v(t) \geq 0$.
2. The time derivative function, $\dot{v}(t) < 0$.
3. The scalar function, $v(t)$, must be radially unbounded ($v(x) \to \infty$ as $x \to \infty$)[13].

The equation given in (2.9) fills the entire above criterion. The purpose of the matrix gain given in equation (2.10) is to determine how quickly the parameter estimate will converge to their steady state value. Generally it is better for the parameter estimates to converge as quickly as possible, however, power and thrust constraints must be observed. A balance between parameter update and power constraints must be found, and for this project a value of one for each scalar gain γ was used.

$$v\left(t\right) = \frac{1}{2}r^T r + \frac{1}{2}\tilde{\theta}^T\Gamma^{-1}\tilde{\theta} \tag{2.9}$$

$$\Gamma = \begin{bmatrix} \gamma_1 & 0 & 0 \\ 0 & \gamma_2 & 0 \\ 0 & 0 & \gamma_3 \end{bmatrix} \tag{2.10}$$

After applying the properties given in (2.11) and (2.13), and realizing that a Lyapunov function is a scalar, the equation given in (2.12) can be manipulated to the form given in

equation (2.14). This form then has equation (2.7) substituted into it to yield (2.15) and after some more manipulation, the form of equation (2.17) is obtained. This equation is needed so that $\dot{\tilde{\theta}}$ can be isolated by setting the terms in parenthesis to zero.

(2.11) $$\frac{d}{dt}\left(\frac{1}{2}r^T r\right) = r^T \dot{r}$$

(2.12) $$\dot{v}(t) = r^T \dot{r} + \left[\frac{1}{2}\dot{\tilde{\theta}}^T \Gamma^{-1}\tilde{\theta}\right] + \left[\frac{1}{2}\tilde{\theta}^T \Gamma^{-1}\dot{\tilde{\theta}}\right]$$

(2.13) $$\frac{1}{2}\dot{\tilde{\theta}}^T \Gamma^{-1}\tilde{\theta} = \frac{1}{2}\tilde{\theta}^T \Gamma^{-1}\dot{\tilde{\theta}}$$

(2.14) $$\dot{v}(t) = r^T \dot{r} + \tilde{\theta}^T \Gamma^{-1}\dot{\tilde{\theta}}$$

(2.15) $$\dot{v}(t) = r^T \left(RM^{-1}y(\dot{P})\tilde{\theta}\right) - r^T K_r r + \tilde{\theta}^T \Gamma^{-1}\dot{\tilde{\theta}}$$

(2.16) $$\dot{v}(t) = -r^T K_r r + r^T RM^{-1}y(\dot{P})\tilde{\theta} + \dot{\tilde{\theta}}^T \Gamma^{-1}\tilde{\theta}$$

(2.17) $$\dot{v}(t) = -r^T K_r r + \left(r^T RM^{-1}y(\dot{P}) + \dot{\tilde{\theta}}^T \Gamma^{-1}\right)\tilde{\theta}$$

Once the terms in parenthesis of (2.17) are set to zero and solved for $\dot{\tilde{\theta}}$ the result of (2.18) is obtained. Once realizing that according to the definition of adaptive control the parameter must be *constant* and *time-invariant*, equation (2.18) resolves to equation (2.20) because $\dot{\theta} = 0$. According to equation (2.19), if $\dot{\theta} = 0$ then $\dot{\tilde{\theta}} = -\dot{\hat{\theta}}$. To obtain the final form of the adaptive update law, equation (2.20) is integrated to produce (2.21).

(2.18) $$\dot{\tilde{\theta}} = \left[-r^T RM^{-1}y(\dot{P})\Gamma\right]^T$$

(2.19) $$\dot{\tilde{\theta}} = \dot{\theta} - \dot{\hat{\theta}}$$

(2.20) $$\dot{\hat{\theta}} = \left[r^T RM^{-1}y(\dot{P})\Gamma\right]^T$$

(2.21) $$\hat{\theta} = \int_0^t \left[r^T RM^{-1}y(\dot{P})\Gamma\right]^T dt$$

(2.22) $$\dot{r} = -K_r r$$

(2.23) $$r(t) = r_0 e^{-K_r t}$$

After substituting the parameter estimate in equation (2.21) into the filtered tracking error derivative of (2.7), the error derivative resolves to (2.22) when the parameter estimate reaches its steady-state value. It is important to notice that the solution of equation (2.22) fits the criteria of a globally asymptotically stable equation (G.A.S.) as given in equation (2.23). A G.A.S. equation is driven to zero in an asymptotic fashion, meaning that the filtered tracking error r is driven to zero in an asymptotic fashion. Based on the definition of the filtered tracking error, the position and orientation errors are also driven to zero in an asymptotic fashion. This means that based on the structure of the controller and the adaptive update law, the vessel is driven to the desired position and orientation in an exponential fashion for all points in the workspace.

9.2 Controller II Simulation Results

The controller that was derived in 9.1 was entered in the modified shell code explained in 8.1. The existing structures for the integrator, differentiator, and timer were expanded upon to support the newly introduced variables for the adaptive update law. The files used to implement Controller II are attached in enclosure 15.2.1. The simulation was run and the path taken by the vehicle was plotted as shown in Figure 7. The path taken by the vessel is almost exactly the same as the simulated results for Controller I. This is expected, because the gain on the adaptive update law was set high and the initial conditions for the drag parameters were very close to their actual values. The simulation was then run again using lower adaptive update gains and initial conditions further away from the actual values. As expected, the performance of the controller suffered and time taken to reach the desired endpoint was greater. The plot from this experiment is not included for the sake of brevity.

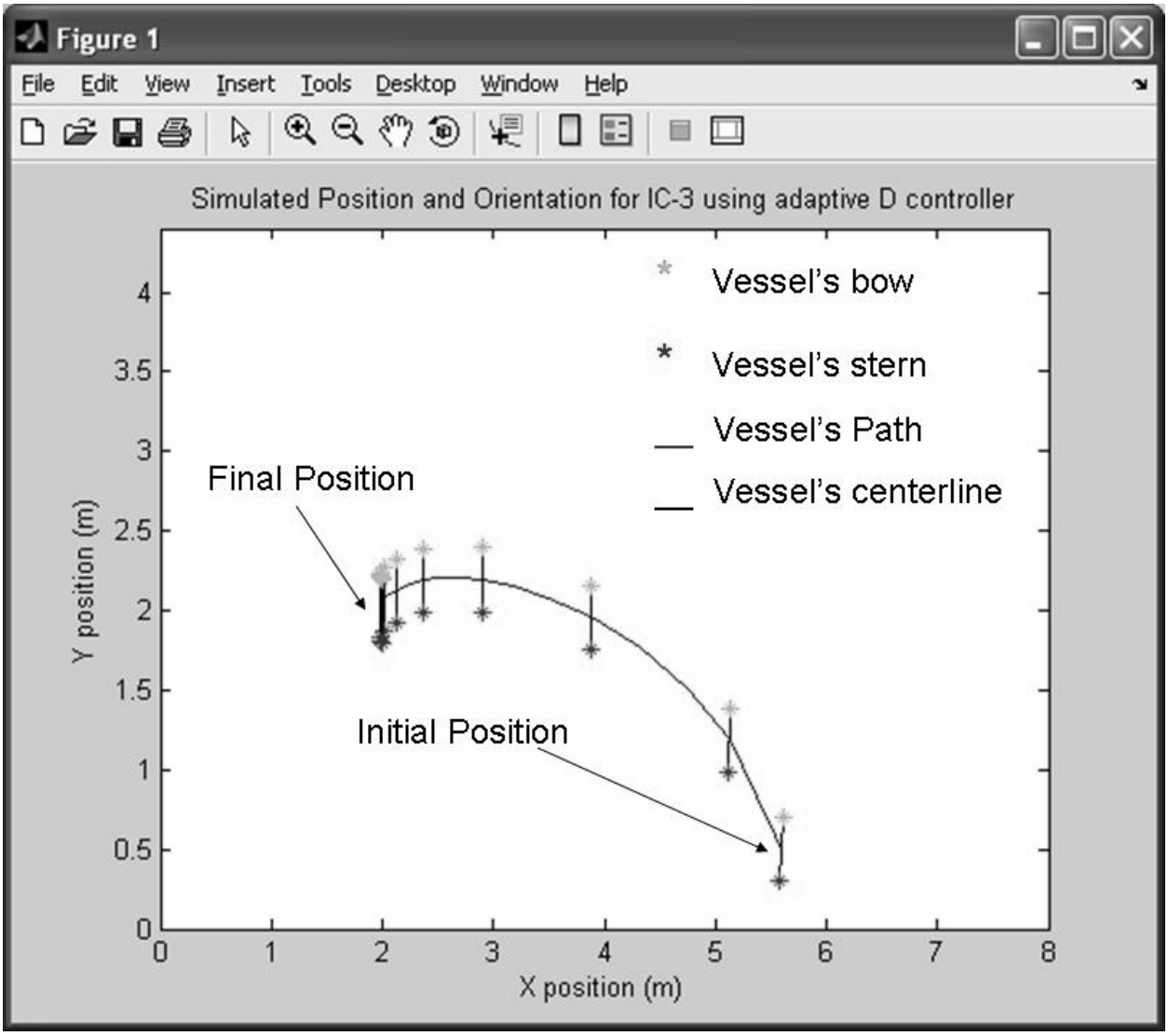


Figure 7: Adaptive Drag Controller Simulation Results for IC-3

The author also experimented with the use of different gain values. There were gain values on the filtered tracking error, positional error, and adaptive update laws. As expected, higher gain values on the filtered tracking error caused the system to react quicker. Higher gain values on the positional error gains caused the system's location to affect the control more than the system's velocity. Higher gain values on the adaptive update laws caused the system parameters to converge to their steady-state values quicker. These gain values required the system to use higher control inputs to affect the motion of the system. However, gain values

ceased to affect the motion of the system over a certain value. This value was usually around ten. Any number greater than ten generally did not cause the system to move any faster. This was due to actuator saturation, which was included in the simulation. The actuators affecting the motion of the system could only nominally produce around two Newtons of thrust. This effectively turned the control algorithm into a bang-bang configuration. Bang-bang controllers essentially have one level of input and they turn on or off to affect the output. They are a very rudimentary form a control; therefore, high control gains deteriorated the system's performance.

Figure 8 shows the drag parameter values verse time for the simulation results of Controller II. For this simulation, the initial values of D_1, D_2, and D_3 were chosen to be 6, 0.5, and 0.1 (Kg/s), respectively to show that there values do not have to converge to the actual values in order for the controller to converge to the desired point. The actual drag values given in the simulation were D_1, D_2, D_3 = 0.05, 0.05, and 0.15 (Kg/s), respectively. As shown in the figure below, the parameter estimates never converge to the actual values. This is due to the condition of persistent excitation that was explained in Section 7.5. The system did not provide a constantly changing input to the controller; therefore, the controller did not need to determine the parameters' actual values in order to drive the filtered tracking error to zero. In fact, the D_1 parameter says close to its initial value of 6 (Kg/s), when its actual value is 0.05 (Kg/s). As shown by the parameter estimates, it takes the system around ten seconds to converge to the desired position, and this can be determined by the amount of time it takes for the parameter estimates to reach their final values.

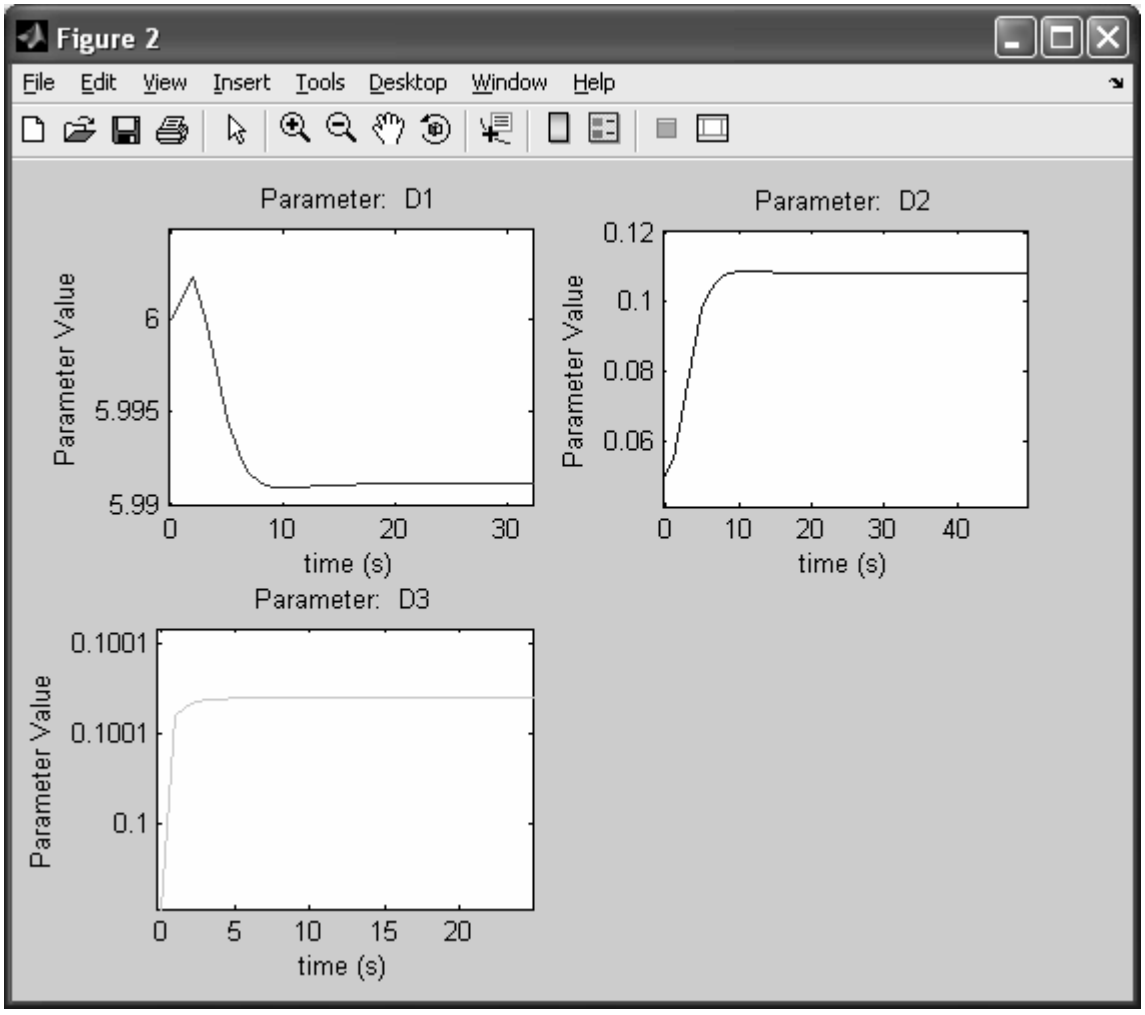


Figure 8: Adaptive Drag Controller Simulation Parameters for IC-3

9.3 Controller II Experimental Results

Figure 9 shows the path taken by the experimental vessel using Controller II. Code used to implement Controller II is shown in enclosure 15.2.2. It is interesting to notice that this path is almost exactly the same as the path taken by Controller I. An inherent advantage that Controller II has during experimentation is the fact the drag estimates used in Controller I are not very accurate. These drag terms are based on numbers obtained by the U.S. Naval Academy's Hydromechanics Laboratory for the particular hull form used for the small-scale experimental vessel. While determining the drag of the model, the Hydromechanics Laboratory only towed

the model bow first, without the simulated tugboats, and without the weight distribution of the vessel's internals. All of these factors have a significant effect on the hydrodynamics of the vessel, and despite the assumption made in 6.1 that the simulated tugboats have no effect of the drag of the system they truly have a great effect of the drag. More accurate modeling of the hydrodynamic properties of the system will be deferred to future work.

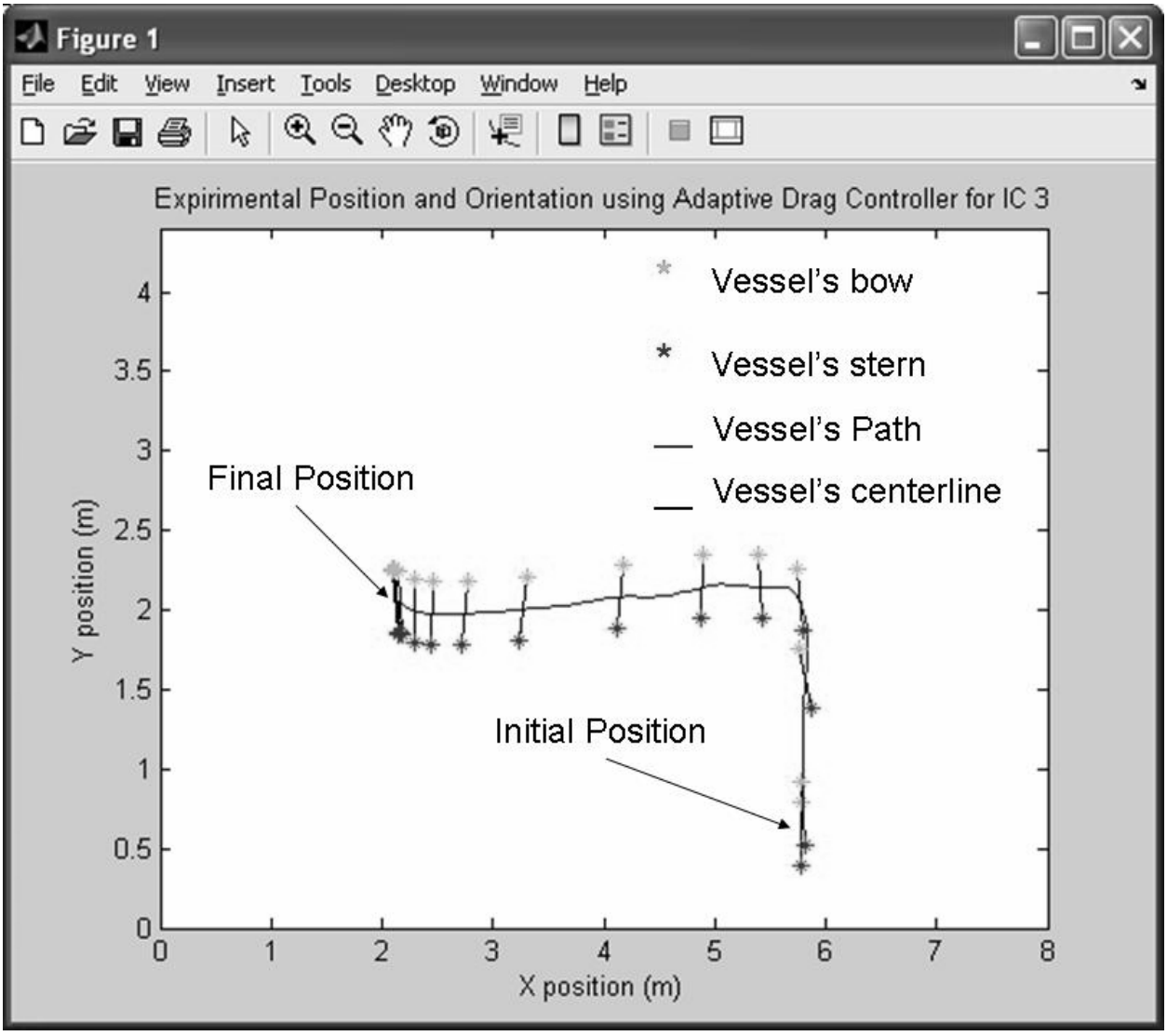


Figure 9: Adaptive Drag Controller Experimental Results for IC-3

Figure 10 shows the drag parameter estimates over the entire experimental run. These graphs show that the drag parameters are actively being updated until the end of the run. After

looking at D_3 it is easy to see that the parameter estimate has a slight oscillation. This slight oscillation can also be seen in orientation history of Figure 9. Both D_1 and D_2 reach their settling value rather quickly, but D_3 looks as though it is still in the settling process as the experimental run is ended. As expected, these experimental drag parameters are different than the simulated drag parameters. Also, the D_3 parameter seems to change while the x and y position changes. This shows that the movements in the x, y, and ψ directions have some coupling, although this coupling is very small and it appears that the assumption that movement is decoupled is true.

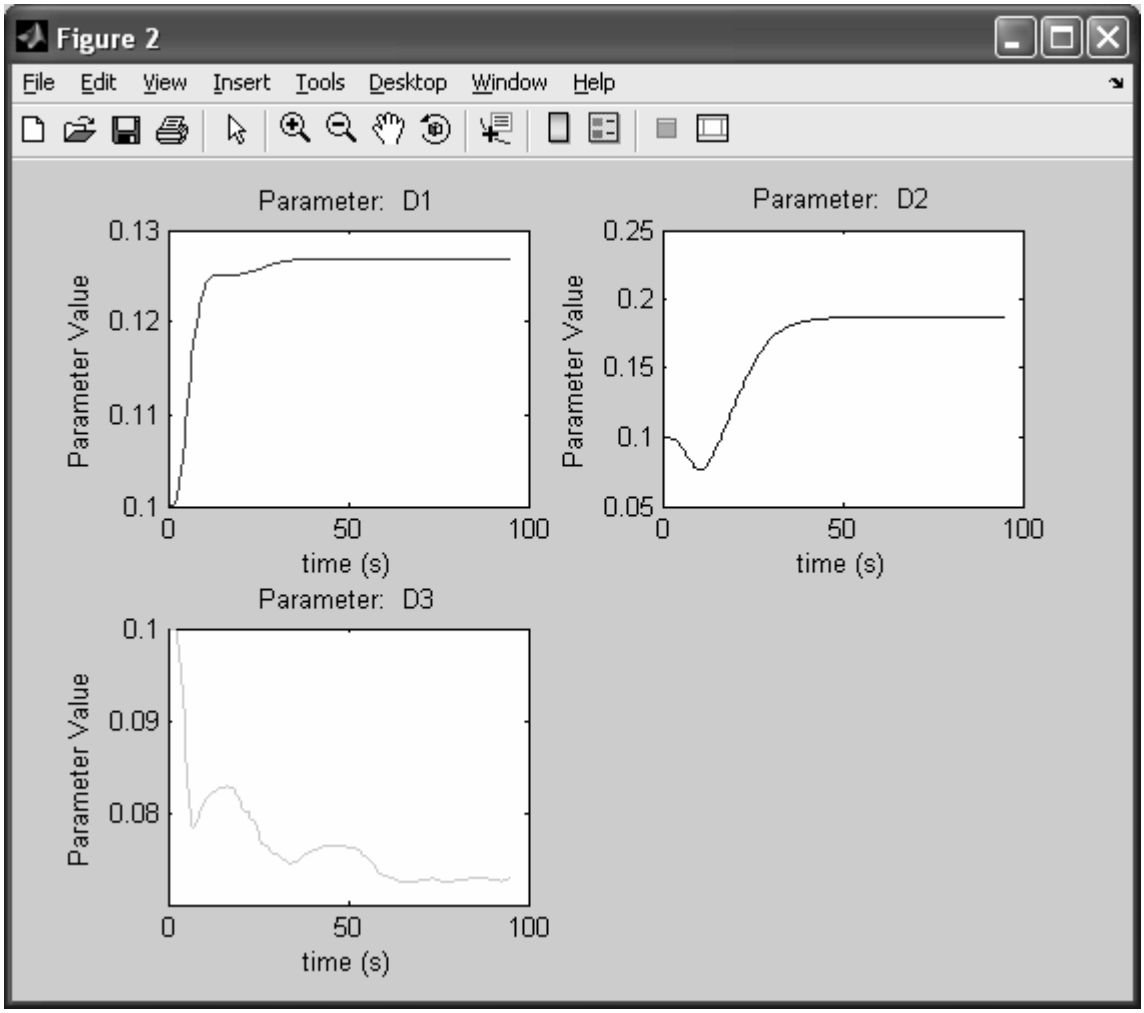


Figure 10: Adaptive Drag Controller Experimental Parameters for IC-3

10. Controller solving Scenario III (unknown tugboat locations)

As stated above in 6.1, Controller III solves the Scenario in which exact tugboat placement is unknown. Although exact tugboat placement is unknown in Controller III, there must be some knowledge of the system, namely the signs of each element in the thrust configuration matrix B. These signs must be known so that B is full rank, meaning that its inverse can be taken. If B is not full rank, its inverse will yield a singularity and the controller will drive the system unstable. To ensure that B is of full rank, its moving parameters must be bounded so their estimates do not pass through zero. If the estimates pass through zero or change their sign, the B matrix will yield a singularity during inversion. Essentially, parameter estimate signs must be known and the adaptive controller will only vary the magnitude of each parameter.

10.1 Controller III Derivation and Proof

To derive a suitable controller for Scenario III, the open-loop filtered tracking error dynamics was used as a starting point and the terminology given in Table 5 was used. To find the proper form for the equation for the manipulation needed to derive the adaptive update law, the author used equation (3.1) and added a virtual control input as shown in equation (3.2). Adding a virtual control input is essentially adding zero to the right hand side of the equation. This form is needed to continue with the derivation. Next, the author must solve for the input in terms of the system's dynamic properties. This is done by solving the open-loop system model for thrust; however, to obtain the proper controller, one must ignore the $-\left[RM^{-1}\hat{B}U\right]$ term on the right hand side of the equation. This can be done because the author will assume perfect knowledge of the $\hat{B}$ parameter for the moment. This term will be accounted for during the

derivation of the adaptive update laws. After solving for thrust while ignoring the $-\left[RM^{-1}\hat{B}U\right]$ term, the controller is obtained in (3.4).

(3.1) $$\dot{r} = \ddot{P}_d + \alpha\dot{e} + \dot{\psi} \times \dot{P} + RM^{-1}DR^T\dot{P} - RM^{-1}BU$$

(3.2) $$\dot{r} = \ddot{P}_d + \alpha\dot{e} + \dot{\psi} \times \dot{P} + RM^{-1}DR^T\dot{P} - RM^{-1}BU + \left[RM^{-1}\hat{B}U - RM^{-1}\hat{B}U\right]$$

Variable	Definition	Variable	Definition	Variable	Definition	Variable	Definition
M [3x3]	Mass of the vessel, Includes added mass and moment of inertia	e [3x1]	Positional error	ψ [1x1]	Yaw angle (measures orientation)	α [3x3]	Position gain
$\dot{v}$ [3x1]	Vessel acceleration	$\dot{P}_d$ [3x1]	Desired velocity	$\dot{\psi}$ [1x1]	Yaw rate	$v(t)$ [1x1]	Lyapunov function
M [1x1]	Vessel's mass	$\ddot{P}_d$ [3x1]	Desired acceleration	r [3x1]	Filtered tracking error	$\dot{r}$ [3x1]	Filtered tracking error derivative
D [3x3]	Hydrodynamic drag	$\dot{e}$ [3x1]	Velocity error	$\dot{P}$ [3x1]	Global velocity	$\dot{V}(t)$ [1x1]	Lyapunov function derivative
v [3x1]	Vessel velocity	P_d [3x1]	Desired Position	k [1x1]	Gain term	I [3x3]	Identity matrix
B [3x3]	Vessel thruster configuration	R [3x3]	Rotation matrix (converts from global to body frame)	P [3x1]	Global position	R^T [3x3]	Transpose of the rotation matrix
U_s [3x1]	Thrust magnitude	K_r [3x3]	Error Gain	B_s [3x3]	Defined swarm thruster configuration	$y(\theta)$ [9x9]	Parameter regression matrix
x [1x1]	X position	y [1x1]	Y position	$u_{a,b,c}$ [1x1]	Scalar member of the U_s matrix	θ [9x1]	Parameter vector
$\dot{x}$ [1x1]	X velocity	$\dot{y}$ [1x1]	Y velocity	$\hat{D}$ [1x1]	Drag parameter estimate	$\hat{\theta}$ [9x1]	Parameter estimate vector
$\tilde{\theta}$ [9x1]	Parameter error vector	$\lvert\hat{b}_{x,y}\rvert$ [1x1]	Absolute value of the scalar parameter estimate	$\dot{\tilde{\theta}}$ [9x1]	Parameter error vector derivative	$\dot{\hat{\theta}}$ [9x1]	Parameter estimate vector derivative
$b_{x,y}$ [1x1]	Scalar member of B matrix	j [1x1]	Vessel's moment of inertia	$\hat{B}$ [3x3]	Redefinition of parameter matrix	$\mathrm{sgn}(b_{x,y})$ [1x1]	Sign of $b_{x,y}$

Table 5: Variables used for the derivation of Controller III.

$$RM^{-1}\hat{B}U = \begin{bmatrix} \cos(\psi) & \sin(\psi) & 0 \\ -\sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{1}{m} & 0 & 0 \\ 0 & \frac{1}{m} & 0 \\ 0 & 0 & \frac{1}{j} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix} \begin{bmatrix} u_a \\ u_b \\ u_c \end{bmatrix} =$$

$$\begin{bmatrix} \frac{1}{m}\left[b_{11}\cos(\psi)u_a + b_{12}\cos(\psi)u_b + b_{13}\cos(\psi)u_c + b_{21}\sin(\psi)u_a + b_{22}\sin(\psi)u_b + b_{23}\sin(\psi)u_c\right] \\ \frac{1}{m}\left[-b_{11}\sin(\psi)u_a - b_{12}\sin(\psi)u_b - b_{13}\sin(\psi)u_c + b_{21}\cos(\psi)u_a + b_{22}\cos(\psi)u_b + b_{23}\cos(\psi)u_c\right] \\ \frac{1}{j}\left[b_{31}u_a + b_{32}u_b + b_{33}u_c\right] \end{bmatrix} =$$

$$\begin{bmatrix} \operatorname{sgn}(b_{11})\frac{\cos(\psi)u_a}{m} & \operatorname{sgn}(b_{12})\frac{\cos(\psi)u_b}{m} & \operatorname{sgn}(b_{13})\frac{\cos(\psi)u_c}{m} & \operatorname{sgn}(b_{21})\frac{\sin(\psi)u_a}{m} & \dots \\ \operatorname{sgn}(b_{11})\frac{-\sin(\psi)u_a}{m} & \operatorname{sgn}(b_{12})\frac{-\sin(\psi)u_b}{m} & \operatorname{sgn}(b_{13})\frac{-\sin(\psi)u_c}{m} & \operatorname{sgn}(b_{21})\frac{\cos(\psi)u_a}{m} & \dots \\ 0 & 0 & 0 & 0 & \dots \\ \end{bmatrix}$$

$$\begin{bmatrix} \operatorname{sgn}(b_{22})\frac{\sin(\psi)u_b}{m} & \operatorname{sgn}(b_{23})\frac{\sin(\psi)u_c}{m} & 0 & 0 & 0 \\ \operatorname{sgn}(b_{22})\frac{\cos(\psi)u_b}{m} & \operatorname{sgn}(b_{23})\frac{\cos(\psi)u_c}{m} & 0 & 0 & 0 \\ 0 & 0 & \operatorname{sgn}(b_{31})\frac{u_a}{j} & \operatorname{sgn}(b_{32})\frac{u_b}{j} & \operatorname{sgn}(b_{33})\frac{u_c}{j} \end{bmatrix} \begin{bmatrix} |\hat{b}_{11}| \\ |\hat{b}_{12}| \\ |\hat{b}_{13}| \\ |\hat{b}_{21}| \\ |\hat{b}_{22}| \\ |\hat{b}_{23}| \\ |\hat{b}_{31}| \\ |\hat{b}_{32}| \\ |\hat{b}_{33}| \end{bmatrix} = y(\theta)\hat{\theta}$$

(3.3)

To start the derivation of an adaptive update law for the thrust input matrix, the elements of the $\hat{B}$ matrix must be parameterized out of the $RM^{-1}\hat{B}U$ term. This long process is shown in (3.3) and the result is the definition of the $y(\theta)\hat{\theta}$ vectors. The $\hat{\theta}$ vector accomplishes the objective set forth in 10. The absolute values of the scalar components of $\hat{B}$ are contained in $\hat{\theta}$. The value

of $\hat{\theta}$ is used to define the scalar members of the actual thrust configuration parameters in $\hat{B}$ as shown in equation (3.5). Equation (3.5) also shows the definition of the filtered tracking error, and this definition is redefined with the error dynamics including the adaptive update parameterization in equation (3.6).

(3.4) $$U_S = \left[R(\psi) M^{-1} \hat{B} \right]^{-1} \left[\ddot{P}_d + \alpha \dot{e} + K_r r + \dot{\psi} \times \dot{P} + R(\psi) M^{-1} D R^T \dot{P} \right]$$

(3.5) $$\hat{B} = \begin{bmatrix} \operatorname{sgn}(b_{11}) \left| \hat{b}_{11} \right| & \cdots & \operatorname{sgn}(b_{13}) \left| \hat{b}_{13} \right| \\ \vdots & \ddots & \vdots \\ \operatorname{sgn}(b_{31}) \left| \hat{b}_{31} \right| & \cdots & \operatorname{sgn}(b_{33}) \left| \hat{b}_{33} \right| \end{bmatrix}, r = \dot{e} + \alpha e$$

(3.6) $$\dot{r} = -k_r r - y(\theta) \tilde{\theta}$$

The definition of $\tilde{\theta}$ is given in (3.7), and this definition is used to define the Lyapunov function given in (3.8). The expression for the filtered tracking error and $\tilde{\theta}$ is substituted into equation (3.8) to yield (3.9). Remember that that one of the central tenants of adaptive control is that the values of the parameters must be constant, therefore $\dot{\tilde{\theta}} = -\dot{\hat{\theta}}$. After this, the Lyapunov function is further manipulated using the properties of scalar derivation to produce the form given in equation (3.11). From this form, the terms in parenthesis are set so zero so that a value for $\dot{\hat{\theta}}$ can be determined. This matrix manipulation process is shown in equations (3.12) and (3.13). After integrating the value for $\dot{\hat{\theta}}$, equation (3.14) is obtained. It is important to notice that once the adaptive update law determines the proper steady state value for $\hat{\theta}$, equation (3.6) resolves to the globally exponential equation given in (3.15).

(3.7) $$\tilde{\theta} = \theta - \hat{\theta}$$

(3.8) $$v(t) = \frac{1}{2} r^T r + \frac{1}{2} \tilde{\theta}^T \tilde{\theta}$$

(3.9) $$\dot{v}(t) = r^T \left[-K_r - y(\theta) \tilde{\theta} \right] + \tilde{\theta}^T (-\dot{\hat{\theta}})$$

(3.10) $$\dot{v}(t) = -K_r r^T r - r^T y(\theta)\tilde{\theta} - \tilde{\theta}^T \dot{\hat{\theta}}$$

(3.11) $$\dot{v}(t) = -K_r r^T r - (r^T y(\tilde{\theta}) - \dot{\hat{\theta}}^T)\tilde{\theta}$$

(3.12) $$\dot{\hat{\theta}}^T = -r^T y(\theta)$$

(3.13) $$\dot{\hat{\theta}}^T = -(r^T y(\theta))^T$$

$$\dot{\hat{\theta}} = -y(\theta)^T r$$

(3.14) $$\hat{\theta} = \int_0^t \left[-y(\theta)^T r \right] dt$$

(3.15) $$\dot{r} = -K_r r$$

10.2 Controller III Simulation Results

The controller that was derived in Section 10.1 was entered in the modified shell code explained in Section 8.1. The existing structures for the integrator, differentiator, and timer were expanded upon to support the newly introduced variables for the adaptive update law. The files used to implement Controller III are attached in enclosure 15.3.1. The simulation was run and the path taken by the vehicle was plotted as shown in Figure 11. Not surprising, the simulated vessel took much the same path as the other two simulations.

The adaptive update parameters contained in $\hat{B}$ are shown in Figure 12. It is important to notice that each of the parameters converges rather quickly to its steady state value. Although each of the parameters does not reach the actual value given in simulation, its steady state value is close to the actual parameter. This could be due to the fact that the author gave the adaptive update laws the actual values as initial conditions. Since the system was not persistently excited, the parameters were not expected to reach their actual values. All of the parameters are the correct signs and their values do not cross zero, therefore, the system is stable.

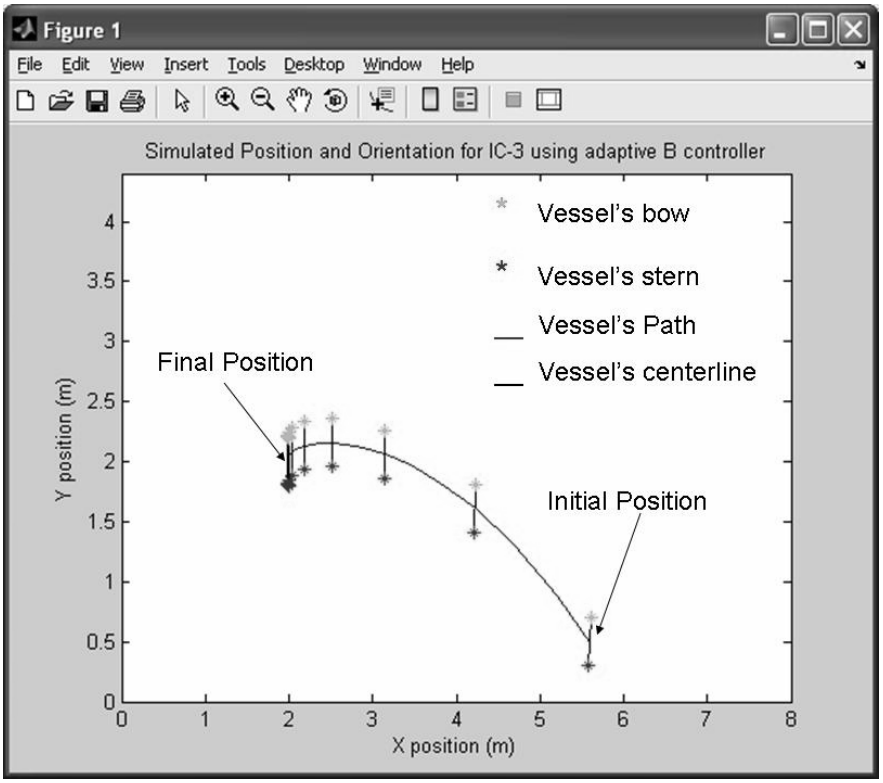


Figure 11: Adaptive B Controller Simulation Results for IC-3

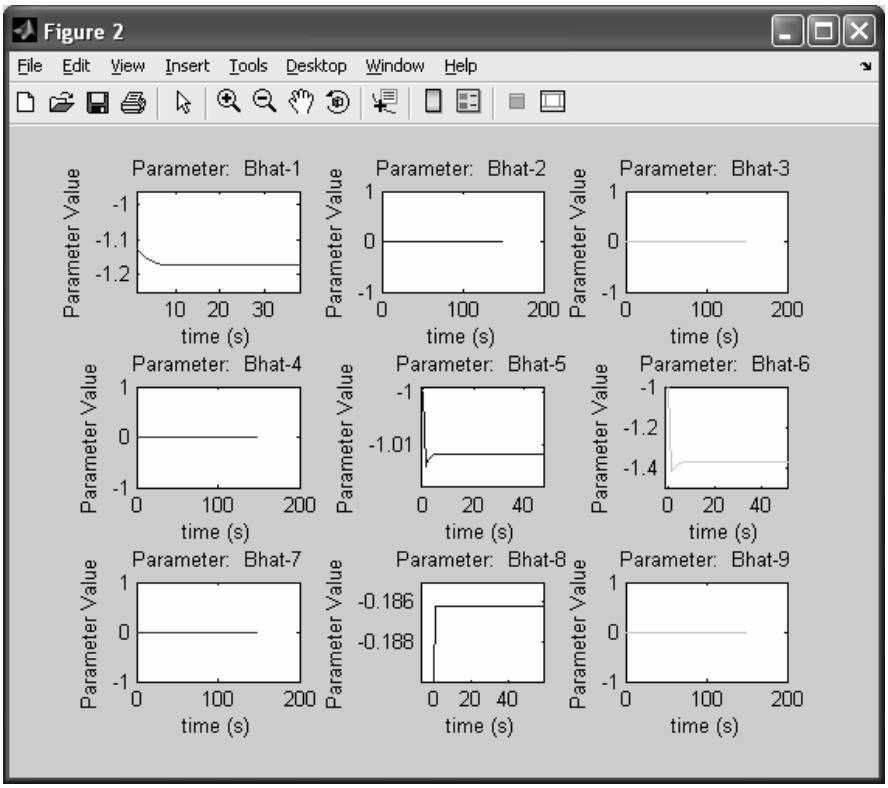


Figure 12: Adaptive B Controller Simulation Parameters

10.3 Controller III Experimental Results

Figure 13 shows the path taken by the experimental vessel using Controller III. It is interesting to notice that this path is very similar to the paths taken by Controller I and Controller II. The code used to implement this controller is in enclosure 15.3.2. This controller does, however, have some oscillation at the end of the experimental run. This oscillation is caused by the updating thrust configuration matrix. The updating thrust configuration matrix affects the whole system, but most notably the rotation of the system. Adapting the magnitude of the elements in the $\hat{B}$ matrix is essentially changing the lever arm of the torque generated in the controller's system dynamics. If the controller believes there is a smaller lever arm then there actually is in the system, it will apply more thrust to accomplish the task then needed. This overshoot in orientation will then feedback to the $\hat{B}$ matrix, and the parameter will be decreased to better model the actual system. This action is shown in the parameter B_5 in Figure 14. These values overshoot their optimal value and then return due to system overshoot.

It is important to notice that as opposed to the simulation, the $\hat{B}$ parameters are still changing at the completion of the run. This is due to the many un-modeled nonlinear forces and coupling affecting the ability of the parameter to converge quickly to its proper steady state value. In fact these nonlinear forces are what cause the oscillation in the parameter value, and this in turn causes a oscillation in the vessel's position. Another factor that could account for the slow changing estimates is the fact that they had to be bound to prevent zero crossing, and thus, an unstable system. In fact, this binding causes B_8 to reach and stay at the tolerance value. This parameter then can not update for the rest of the experimental run, and the other parameters must now compensate for the lost system adaptability.

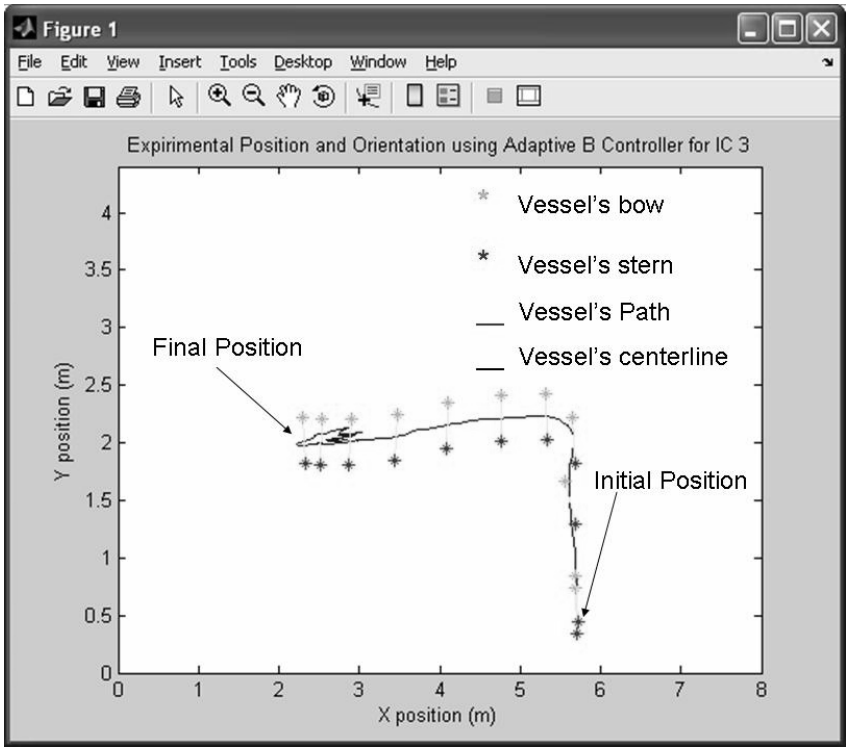


Figure 13: Adaptive B Controller Experimental Results for IC-3

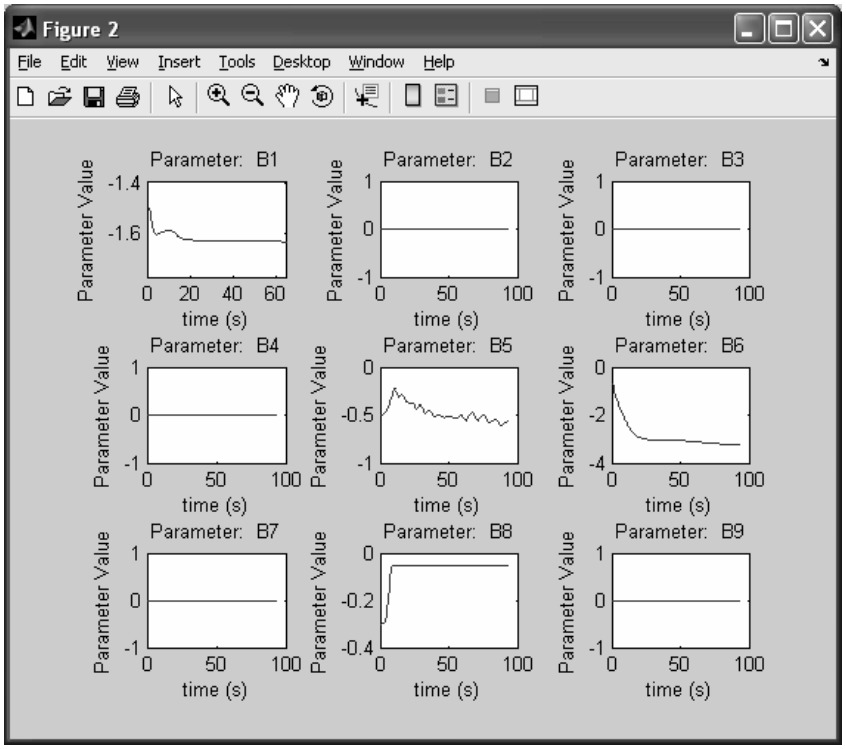


Figure 14: Adaptive B Controller Experimental Parameters

11. Performance Analysis, Independence Analysis, and Controller Comparison

To determine the *performance* of each controller, the experimental data collected was processed using several metrics that were developed to reveal desirable characteristics. Specifically, these characteristics were settling time, positional error, and thrust conservation. Minimal settling time was desired so that experimental vessel could reach the final position and orientation as fast as possible. Minimal positional error was desired so that the experimental vessel would reach the specified endpoint as accurately as possible, while taking the most direct route. Thrust conservation was desired so that the experimental vessel could reach the specified endpoint using the least amount of energy. Each one of the performance metrics used is detailed in the following paragraphs.

11.1 Performance Metrics

11.1.1 Settling Time

Settling time was the first performance metric developed. Settling time was defined as the time the vessel took to reach a certain distance from the final position. The aforementioned distance was determined by taking the normal of the tolerance vector. The tolerance vector included the maximum X-positional error, Y-positional error, and angle error for which the vessel was considered sufficiently close to the desired endpoint. An important detail to note was that the angle error had to be converted from radians to a distance so that units would agree. This was done using the equation $\psi \cdot Radius = Arclength$, where *Radius* is half the length of the vessel. The arc length error was then used in the normal to define the proper tolerance vector. In Table 6, the corresponding tolerance vectors and normal values are displayed.

Maximum error and normal value	Initial Conditions 1	Initial Conditions 2	Initial Conditions 3
X-positional error (m)	0.30	0.11	0.30
Y-positional error (m)	0.30	0.11	0.30
Angle error (m)	0.12	0.12	0.12
Normal (m)	0.44	0.20	0.44

Table 6: Tolerance vector values and corresponding normal values for each set of initial conditions

To determine settling time, the normal of the error vector was determined for each control iteration and then compared to the normal of the tolerance vector. If the value of the error normal was less than the value of the tolerance normal, the vessel had reached a point sufficiently close to the desired endpoint. This did not necessarily mean that the vessel had settled at the desired endpoint as it could oscillate out of the this position, therefore, five consecutive control iterations meeting the above criteria was required. Once this happen, the vessel had settled and the time corresponding to the control iteration was saved. This time was then defined as the settling time. Enclosure 15.4 shows the code used to determine the controller's settling time. Each controller's performance, based on settling time, is given in Table 7.

	Initial Conditions 1	Initial Conditions 2	Initial Conditions 3	Average
Controller I (Full)	53.60	85.72	46.85	62.06
Controller II (Adaptive Drag)	53.27	56.63	49.23	53.04
Controller III (Adaptive B)	95.18	88.70	50.15	78.01

Table 7: Settling Time (seconds) for each Controller.

Table 7 shows a very interesting result. It shows that Controller II has the best performance when using settling time as a metric. Controller II actually settles faster than Controller I in two of the three sets of initial conditions, contrary to what was expected at the beginning of the project. This will be discussed in detail below. As expected, Controller III has the worst performance when settling time is used as a metric.

11.1.2 Positional Error

The next performance metric developed was positional error. Positional error is important because it is a good measure of the efficiency of the path taken to the desired endpoint. The less positional error over the path, the more direct the route taken. The controller is also penalized for any steady state error through this metric, therefore, the lower the positional performance metric, the less total positional error. The positional error metric and settling time metric are closely related in that positional error is actually determined while finding the settling time. The positional error metric is defined in equation (4.1).

$$PM = \int_0^t \|error\|^2 dt \quad \text{Where, } error = \begin{bmatrix} x_d - x \\ y_d - y \\ (\psi_d - \psi) \cdot Radius \end{bmatrix} \tag{4.1}$$

Essentially, the normal of the positional error was calculated and integrated for each control iteration. To find the average positional error, this number was then divided by the number of control iterations. Enclosure 15.4 shows the code written to implement this process. Each controller's performance, based on positional error, is given in Table 8.

	Initial Conditions 1	Initial Conditions 2	Initial Conditions 3	Average
Controller I (Full)	0.83	0.82	1.54	1.06
Controller II (Adaptive Drag)	0.90	0.58	1.71	1.06
Controller III (Adaptive B)	1.09	0.56	1.86	1.17

Table 8: Positional Error (meters) for each Controller.

Table 8 shows that Controllers I and II have the same performance when it comes to positional error. This is interesting as it is contrary to the original hypothesis. Controller I was expected to outperform both Controllers II and III. Reasons for this discrepancy will be discussed below. Controller III has the worst performance, using positional error as a metric. This finding supports the original hypothesis.

11.1.3 Thrust Conservation

The last performance metric developed was thrust conservation. This metric essentially calculates the average thrust used per control iteration. Thrust conservation was used as a performance metric because thrust directly correlates to energy. The less thrust created by the system, the less energy that is used to move the system to the desired endpoint. Minimizing energy is desirable because this conserves battery power and fuel. Although battery power and fuel are not a consideration in this project's experimental vessel, they are very much a consideration in the real world. To calculate the total thrust, equation (4.2) is used.

(4.2) $$TM = \int_0^t (u_1 + u_2 + u_3 + u_4 + u_5 + u_6)dt$$

Equation (4.2) essentially integrates the total thrust used by all tugboats to manipulate the barge. Since all of the thrust values are positive due to the commutation strategy, there is no need to square or take the absolute value. The result is then averaged by dividing it by the number of

control iteration. This gives the average thrust used per control iteration. Enclosure 15.4 shows the code written to implement this process. Each controller's performance, based on thrust conservation, is given in Table 9.

	Initial Conditions 1	Initial Conditions 2	Initial Conditions 3	Average
Controller I (Full)	1.78	1.53	1.26	1.52
Controller II (Adaptive Drag)	1.94	1.34	1.23	1.51
Controller III (Adaptive B)	2.61	1.71	1.62	1.98

Table 9: Thrust Conservation (Newtons) for each Controller.

Table 9 shows that Controller II has the best performance when it comes to thrust conservation. This is contrary to the original hypothesis that Controller I would have the best performance. This discrepancy will be explained below. As expected, Controller III has the worst performance when it comes to thrust conservation. This result supports the original hypothesis.

11.2 Performance Comparison

Using all of the previously defined performance metrics, Controller II has comparable or better performance than Controller I. This is contrary to the original hypothesis that performance would fall in the following order (from best performance to worst performance):

1. Controller I,
2. Controller II,
3. then Controller III.

There are many reasons why this could be true, however, the most viable reason is that Controller I relies on exact knowledge of all the hydrodynamic parameters. Since the objective

of this project was to develop controllers to solve the aforementioned Scenarios and determine their performance and independence, only a rough estimate of the hydrodynamic parameters was calculated. Hydrodynamic drag is inherently difficult to measure for a system of this complexity, which is why its determination was not a goal of this research. This rough calculation may have been what caused the performance of Controller I to be worse than the performance of Controller II. Since Controller I required knowledge of the hydrodynamic drag, and this knowledge was not exact, the controller was not able to correctly cancel out the drag forces. This gave Controller II an inherent advantage, because it did not need exact knowledge of the hydrodynamic drag and was able to adapt its estimate of drag. Controller II was able to correctly cancel the drag forces while Controller I was not able to correctly cancel the drag forces. In this case, freedom from exact model knowledge proved to be a performance advantage.

Controller III, as expected, had worse performance than the other controllers. This was expected because of the nature of the parameters it is updating. Controller III, as explained in Section 10.1, varies members of the $\hat{B}$ matrix. This matrix essentially contains where each of the tugboat thrusts acts on the vessel, in addition to the length of the lever arms for the orientation manipulation. All of these factors have a great influence on the controllability of the vessel. As shown above, Controller III had a much longer settling time and thrust usage and only a slightly large positional error than the other two controllers. The large settling time and thrust usage was due to oscillation induced in the controller as it near the desired location. This oscillation was due to the updating $\hat{B}$ parameters. As the parameters updated, the vessel oscillated its way to the desired position. This oscillation cause the controller to uses much more thrust and caused the settling time to be quite long. The controller may have reached the desired point before the

settling time; however, it oscillated out of the acceptable region soon thereafter causing the controller to have a longer settling time. The positional error was only slightly larger than the other two controllers because the oscillations were relatively small.

11.3 Independence Analysis

To determine the *Independence* of each controller, its derivation was used. Specifically, the derivations in Sections 8.1, 9.1, and 10.1 along with the Scenario derivations from Section 6.3 were used to determine the model knowledge needed by the controller. Model knowledge is the required information, to include hydrodynamic properties and placement of the tugboats. To characterize the *Independence* of a controller, the number of unknown variables will be used. For example, Controller I requires exact model knowledge, therefore, its independence is zero. Controller II does not require exact knowledge of the hydrodynamic drag, which contains three unknown variables. Therefore, Controller II's independence is three. Controller III does not require exact knowledge of the thrust configuration matrix $\hat{B}$, which contains nine unknown variables; therefore, its independence is nine. A synopsis of the controllers' independence is given in Table 10.

	Number of Unknown Variables	Independence metric
Controller I (known positions)	0	0
Controller II (unknown drag)	3	3
Controller III (unknown positions)	9	9

Table 10: Independence metric (number of variables) for each controller

11.4 Controller Comparison

In light of the comparisons already done in Sections 11.2 and 11.3, recommendations will be given below for each controller's use pertaining to *Performance* and *Independence*.

Controller suitability pertains to its application and is best decided on a case by case basis depending on the specific objectives of the situation. In this regard, the suitable controller for a particular application is not always clear cut and is very subjective. Generalizations about the suitability for certain applications will now be given.

In applications such as docking or traversing narrow channels, either Controller I or Controller II would be the most suitable; depending on whether the vessel's exact hydrodynamic drag is known. In real world applications, it is very doubtful that exact information about the drag properties of a vessel would be known, therefore, Controller II is the most suitable. Controller II was chosen for docking application because it offers the ultimate performance with regards to all of the performance metrics. Specifically, the superiority of Controller II's positional error performance is important in docking and narrow channel applications where precision is required.

In applications where the exact placement of tug boats can not be guaranteed, Controller III would yield the best performance. When testing Controller I, the wrong B values were used by accident during one of the testing runs. This controller promptly became unstable due to the fact it was not properly distributing thrust to the correct locations in order to manipulate the barge. When Controllers I and II have incorrect thrust configuration values their performance not only suffers but it can not be guaranteed that the vessel will reach the desired endpoint. However, if Controller III has incorrect magnitudes of the thrust configuration values it can still properly control the vessel. Its performance is not comparable when the other controllers have correct knowledge of the thrust configuration; however, it greatly outperforms the other controllers if the opposite is true.

In general, Controller II had the best mix of performance and independence. Controller II had the best experimental performance overall, and was the second most independent controller. Controller III was the most independent, but in testing it had the worst performance. Controller I was not independent and did not have the best performance. However, it is believed that Controller I had an inherent disadvantage due to the inexact hydrodynamic property values used in this research. This will be resolved in future work by determining the proper hydrodynamic properties and then retesting Controller I. In general, the order of best performance with regards to independence is given below (best performance to worst performance):

1. Controller II,
2. Controller I,
3. then Controller III.

12. **Simulation**

12.1 S-function

Simulation with the Matlab software was used to determine if the derived controllers and adaptive update laws would perform as expected. The main contribution to the simulation facet of the project during the fall semester was developing the framework to simulate subsequently developed controllers. The developed framework used a built-in feature of Matlab called the S-function. The S-function was essentially a Matlab m-file that was built into a loop containing adaptable code to integrating variables. The S-function was very easy to use as compared to its counterpart, Simulink, because if one changed the model or controller of a system, there was no need to reconstruct a Simulink model to take the changes into account. When using an S-function, if the model or controller changes, all one had to do was change two or three lines of code. This code representation of Simulink saved the user hours of time as models and

controllers were revised and simulated continually. An example of the base code of the S-function is given below in Figure 15.

```
 function [sys,x0,str,ts] = shipdynandcontrol(t,x,u,flag)
%switch flag,
   case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
   case 1,
    sys=mdlDerivatives(t,x,u);
  case 2,
    sys=mdlUpdate(t,x,u);
  case 3,
    sys=mdlOutputs(t,x,u);
  case 4,
    sys=mdlGetTimeOfNextVarHit(t,x,u);
  case 9,
    sys=mdlTerminate(t,x,u);
  otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;

sizes.NumContStates  = 4;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 10;
sizes.NumInputs      = 0;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1;   % at least one sample time is needed

function sys=mdlDerivatives(t,x,u)
V = x(1:2,1);
theta_hat = x(3:4,1);
[Vdot,        e,        Vd,        U,        theta_hat_dot]        =
AdaptiveControl_D(t,V,theta_hat);
Xdot = Vdot;
sys  = [Xdot; theta_hat_dot; ];
% end mdlDerivatives
function sys=mdlUpdate(t,x,u)
sys = []
% end mdlUpdate
function sys=mdlOutputs(t,x,u)
V = x(1:2,1);
theta_hat = x(3:4,1);
[Vdot,        e,        Vd,        U,        theta_hat_dot]        =
AdaptiveControl_D(t,V,theta_hat);
sys = [V; e; Vd; U; theta_hat];
```

Figure 15: Base S-function code

In the sample S-function given above, the only code that the user had to change for different simulations was the code highlighted in red and of a larger size. In the first highlighted

section, all the user had to change for different models was the number of discontinuous and continuous states along with the number of inputs and outputs. These quantities were determined by the system model along with the number of variables that needed to be integrated. In the second and third section of red and large code, all the user needed to change was the definition of variables, in this case the first two red lines, and the function AdaptiveControl_D. AdaptiveControl_D was where the user defined the specifications of the system, along with the system model, controller, and adaptive update law. This function was easy to change, and while simulating the different Scenarios with the experimental apparatus as the model, all the user had to change for the different Scenarios was the lines of code containing the equation of the controller and the equation of the adaptive update laws.

13. Experimental Vessel Design and Construction

13.1 Vessel Design

Experimental vessel design had somewhat evolved from what was previously proposed. Figure 16 showed the proposed design of the small scale experimental vessel. Essentially, the proposed vessel consisted of a hull made of closed cell foam, 6-7 variable angle thruster pods, and electronics including an on-board computer and wireless modem. After various modifications to the original design, the small scale experimental vessel took shape to what is shown in Figure 17 and Figure 18. Modifications to the original design included: using a prefabricated hull rather than closed cell foam and using fixed angle thruster pods rather than variable angle thruster pods. The reasons for these deviations are given in the following paragraphs.

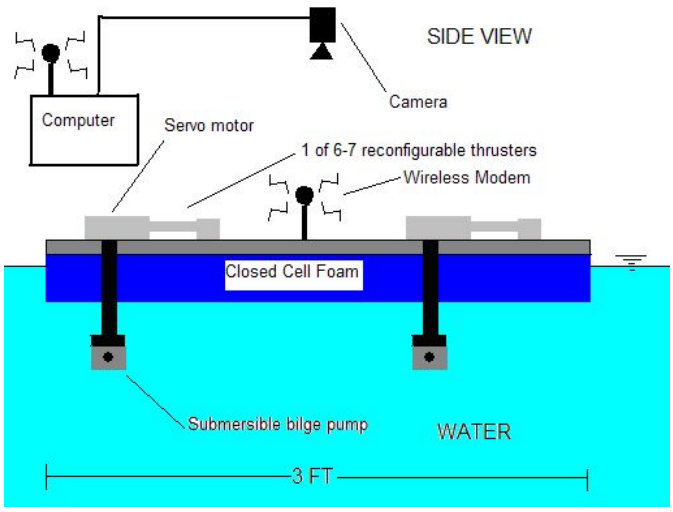


Figure 16: Original Vessel Schematic

Hull design was changed based on the convenience and availability of a prefabricated hull. The original design used closed cell foam because of its price, availability, and buoyancy. However, using closed cell foam has disadvantages, including unrealistic hydrodynamic properties and issues pertaining to durability. Originally, the benefits of using closed cell foam for hull fabrication outweighed the disadvantages. However, after discovering a prefabricated fiberglass YP model hull in the lab, the author decided to use it rather than closed cell foam. This fiberglass hull had all the benefits of closed cell foam but also compensated for its disadvantages. The fiberglass hull has hydrodynamic properties similar to an actual vessel, a YP, and the author obtained the value of these properties from the Oceanography Department. The fiberglass hull was also very durable, as the particular hull had been used many times before and was approximately 15 years old. This adaptation to the vessel had vastly improved its durability, and modeled a real world vessel more closely than closed cell foam.

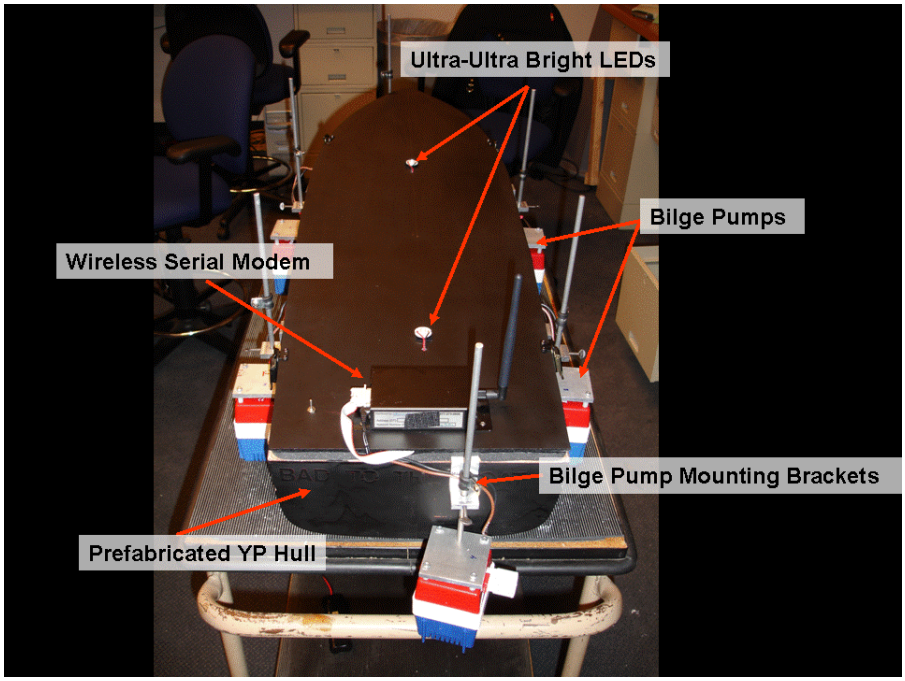


Figure 17: Experimental Vessel (Top)

Fixed angle thruster pod design was used rather than variable angle thruster pod design due to the nature of the control strategy that will be implemented on the vessel. Currently, there exists a control strategy using fixed angles in the previous work [1]. This strategy was adapted to fit the other design Scenarios. Due to the fixed angle control strategy, the author designed the bilge pump mounting brackets shown in Figure 17 and Figure 18. These brackets were then constructed by the machine shop. Although the bilge pump mounting brackets currently have a fixed angle, they are adaptable if one wished to implement a variable angle control design strategy in the future. Changes to implement a variable angle design would include attaching a hobby servo and gear assembly to each bracket. The internals of the vessel already have all the hardware necessary to control the hobby servos.

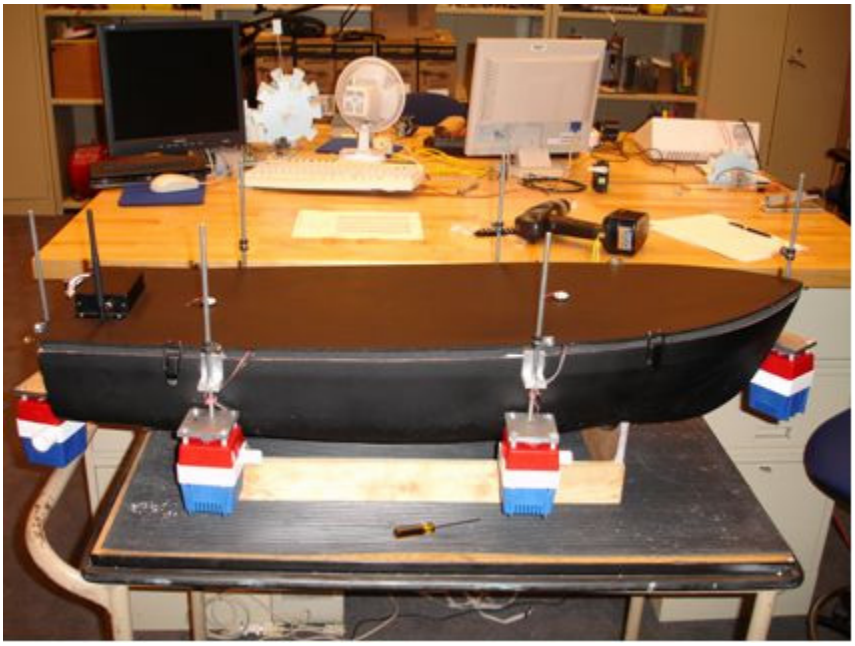


Figure 18: Experimental Vessel (Side)

13.2 Vessel Internals

The vessel's internals consisted of batteries and the control board connected with the corresponding wires and cables.

13.2.1 Batteries

The batteries were two 12V Powersonic lead-acid batteries connected in parallel that provide power for the bilge pumps and various peripheral devices and one 7.2V nickel cadmium battery pack that provided power for low power devices such as the SV203 boards. These batteries were connected to the control board through switches, fuses, and supply terminals as shown in Figure 19. Power for each device was then pulled off of the supply terminals (6V, 12V, and Ground) and run through wire.

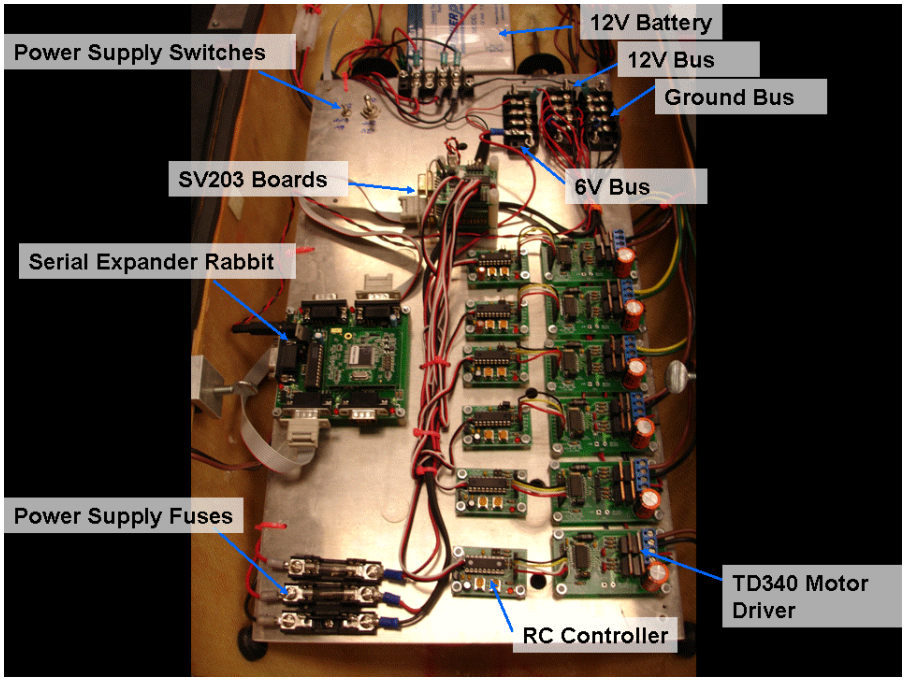


Figure 19: "Control Board" of the Vessel

13.2.2 Control Board

The majority of the electronics internal to the vessel were mounted on an aluminum sheet and this assembly was nicknamed the "control board." Specifically, the control board consisted of the Serial Expander Rabbit on-board computer, two daisy-chained SV203 boards, 6 RC controller boards, and 6 TD340 motor driver boards as shown in Figure 20. All of these parts were constructed by the WSE TSD department. The author's contribution to the construction of the control board was selecting, mounting, and integrating the parts. The purpose of each part is detailed below:

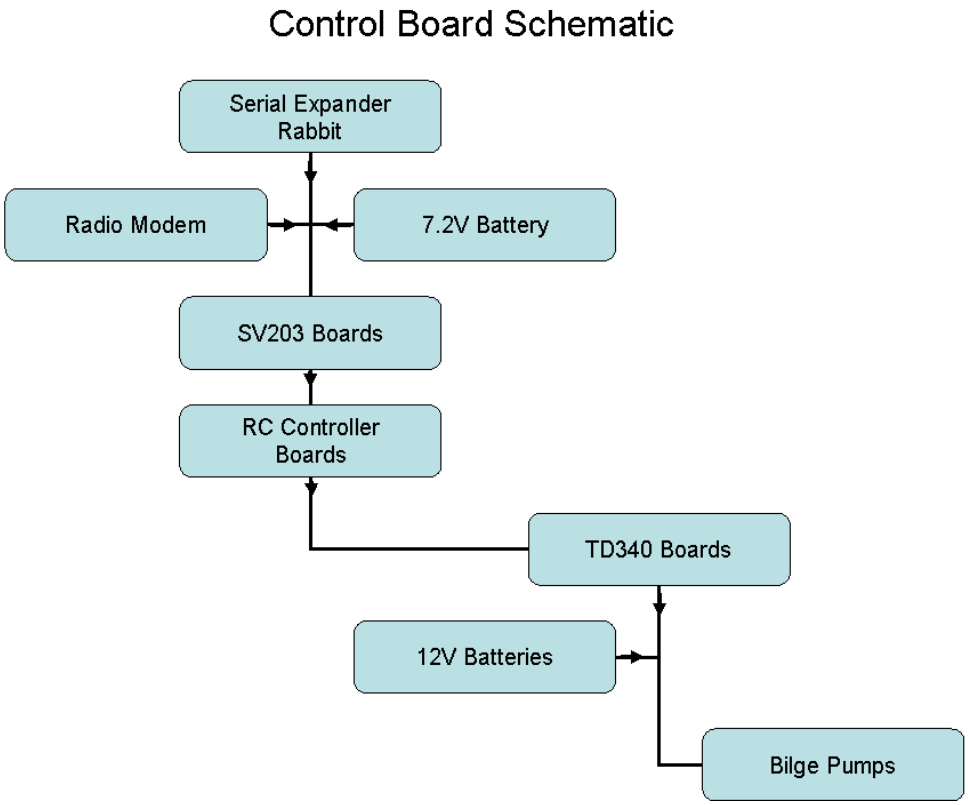


Figure 20: Control Board Schematic

1. Serial Expander Rabbit: The Rabbit microcontroller was a small computer that is mounted on-board the vessel. The Rabbit was needed to receive the proper thrust commands from the base station and distribute them to the proper tug boat. The Serial Expander Rabbit was a special version of the System Engineering Department's single board computer mainstay, the Rabbit microcontroller. This microcontroller consisted of the Rabbit 3000 microprocessor and 5 serial ports. This microcontroller was different than other Rabbit microcontrollers because it does not include peripherals such as A/D converts, D/A converts, etc. Instead of these devices, the Serial Expander Rabbit included 4 more serial ports which make it ideal for the project. The author did not need the aforementioned peripherals; however, the project was serial

communications intensive. This board provided needed serial ports without unused peripherals.

2. SV203 Boards: The purpose of the SV203 board was to provide a pulse width modulated (PWM) signal when given an input in the range of 1 to 255 servo counts. This signal could be used to move a servo motor to the correct position or could be converted and amplified to serve as a throttle for a motor. The input was in string form, which was essentially a sentence consisting of ASCII characters. The program on the SV203 board's embedded PIC processor decoded the input string and then sent a corresponding PWM signal to the specified output port.
3. RC Controller Boards: The purpose of the RC controller board was to convert the position PWM signal sent out from the SV203 board to a continuous speed signal that could be used to control a motor driver. This was done by the code on the board's embedded PIC processor. The output of the RC controller board was another PWM signal; however, this signal was continuous and held until a new input was received by the board.
4. TD340 Boards: The purpose of the TD340 motor drive board was to take the PWM speed signal from the RC controller board and convert that signal to a DC voltage capable of driving a motor. This DC voltage level then corresponded to motor speed. Once again, low power signal (control signal) conversion was done by the code on the board's embedded PIC processor. This control signal then served as an input to the four operation amplifiers mounted on the board. These op-amps then magnified the signal to the correct DC voltage, and this was then supply to the motor. Actual motor speed for this DC voltage level varied based on the characteristic of the motor.

However, for the system, this DC voltage level gave approximately the same motor speed for each bilge pump due to their similarity.

13.3 Vision System

The vision system of the experimental vessel was analogous to the GPS and compass on an actual vessel. This system was used to determine the experimental vessel's position and orientation in the workspace. The vision system consisted of the following items: two high intensity LEDs of different color, a wide field-of-view webcam, a laptop computer running code using Matlab's image acquisition and processing toolboxes, and two serial modems that transmitted the position and orientation to the experimental vessel's onboard computer. The operation of the vision system will be described in the following paragraphs.

Orientation and position were obtained by tracking two LEDs, mounted on the top of the experimental vessel, with a webcam. These LEDs were special-ordered because of their intensity and wide viewing angle and they are shown in Figure 21. Each LED consumed a watt of power and had a 70 degree field-of-view. This allowed the webcam to see the LED even if it was not directly below. LEDs were chosen as tracking objects due to their light invariance. Tracking the color of objects that did not produce their own light was dependent on the ambient light in the room. To increase the reliability of the vision system, the author tracked these different colored LEDs in a nearly dark environment to promote color invariance. The author painted the hull of the boat a flat black to cut down on glare from the LEDs and to also to ensure the boat blended into the background during low light conditions. This also made sure that the camera was only tracking the center of each LED, not reflections off the vessel's cover or the surrounding water by ensuring that potential tracking objects contained a certain number of contiguous pixels.

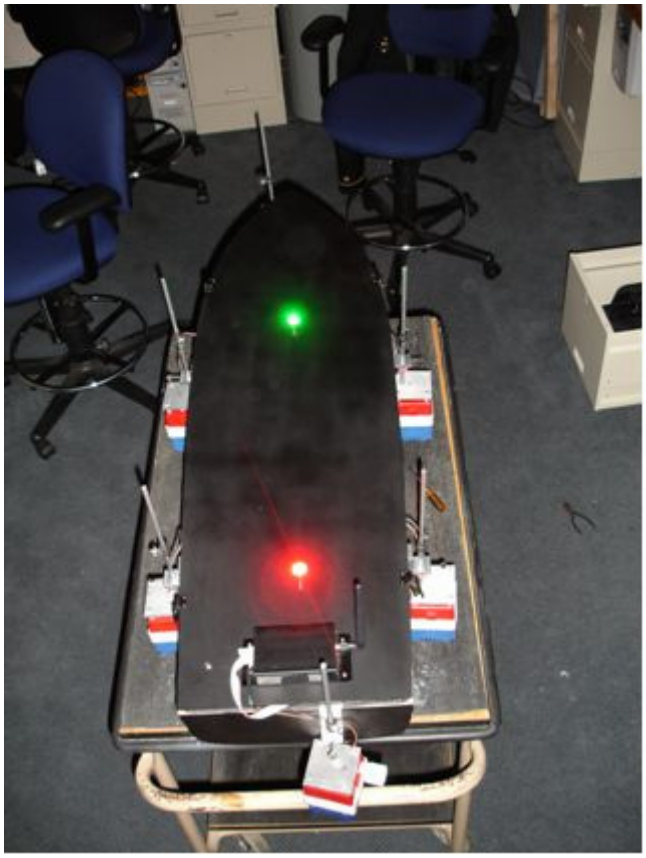


Figure 21: Vessel LEDs used for Light Invariant Tracking

The light from these two LEDs was then captured by the wide field-of-view webcam that was mounted over the experimental vessel. A wide field-of-view camera was used to increase the size of the area in which the vessel could operate since the ceiling height was fixed. The image captured by the webcam was then transmitted to a laptop computer running Matlab code that used the image processing and acquisition toolboxes to segment and identify each LED. This new image only showed the binary image containing the light from the two LEDs and this was used to compute the vessel's position and orientation. Thresholding consisted of running each pixel in the image through a series of conditional statements for each primary color. If the pixel met the criteria of the conditional statement, then it was saved in a binary image, an image showing the pixels that met the criteria in white and every other pixel in black. The position and

orientation was sent to the vessel's on-board computer through a pair (send and receive) wireless serial modems. Once the message containing the vessel's position and orientation was received, the on-board computer read and stored these values for use by the control code.

13.4 System Integration

After constructing the physical components of the experimental vessel, the author had to integrate the system. This partly consisted of getting each physical and electronic part of the vessel to work together through communication. There were two ways that the components talked to each other, either they sent an electronic signal, such as a PWM signal, through wire or they sent a serial communication either through wire or wirelessly. All of the electronic signal communications were preprogrammed by TSD, so the only facet of communication the author had to integrate was serial communication. The other aspect of system integration was developing shell code in which to run the developed control algorithms. The purpose of the shell code was to act as an online integrator, clock, and communication hub. The project's work in serial communications and shell code will be detailed below.

13.4.1 Serial Communications

Serial Communications were used to transmit the vessel its orientation and position from the vision system and were also used to communicate between the vessel's on-board computer and the motor control electronics. A serial communication is essentially a message that was passed over wire sequentially. There was typically one wire for transmitting and one wire for receiving between the two devices that are communicating to each other. Problems inherent to serial communications will be detailed in the following paragraphs.

Difficulty with hardware implementation included problems with buffers and serial cables. All devices that have a serial port also have a buffer, which is a memory location to store

incoming and outgoing messages. This buffer needed to be cleared before each communication session to ensure that information left in the buffer was not being sent. If the buffer was not cleared, the whole communication would be corrupted. The author ran into this problem while originally trying to send a serial communication. This problem was solved by including clear commands in the code. Another hardware problem encountered was using the correct serial cable between devices. Apparently there are two cables used in serial communications, a one-to-one cable that is used to communicate between a computer and a peripheral device and a cable that had the send and receive wires crossed to communicate between computers. A one-to-one cable was used between a computer and a peripheral device because the peripheral device already crossed the send and receive wires within its hardware. This was a problem when trying to use the serial hyperlink on a desktop computer to determine what the vessel's onboard computer was receiving from the laptop. The author solved this problem by constructing a wire that had the send and receive wires crossed to communicate between the two computers.

Software problems in serial communications mainly consisted of timing problems and message passing problems. Timing was very important in serial communications because both the sending and receiving devices must have been coordinated to pass the proper message. If the sending or receiving device stopped sending or receiving in the middle of a message, the message would not be passed in its entirety, therefore, making it useless. It was important to remember that, in the project, both the sending and receiving devices were computers that also had other tasks. These other tasks could interrupt the message passing and inadvertently corrupt the message, and the communications program took this into account by using a handshake protocol. This protocol communicated between devices to make sure that each was ready to pass the message. Once the message was passed it was sent again to tell the computers that it was

safe to go to other tasks. This ensured that the message was sent in its entirety; however, it did not make sure the correct message was being passed. Stopping extra characters from being sent had consumed a good amount of time. For example, Matlab's serial communication send command automatically appended a new line feed character. The experimental vessel was receiving the correct message; however, this message was always preceded by the new line feed character. It took some time to discover that this was an inherent feature of the Matlab command as it was not documented. The lesson learned from this problem was that one must ensure that the correct message was being passed between devices, because it was very easy for an extra character to be sent. Since serial communications is sequential, this corrupted every following message.

13.4.2 Shell Code

The author's major contribution to system integration fall semester had been to develop shell code from which the control algorithm would be implemented. Using the shell code, the code for each new Scenario was just a revision of a few lines to incorporate the new controller. Specifically, the author had completed the code for receiving a serial communication from the vision system and code for sending the proper motor command to the peripheral motor throttle. Other work done on the shell code during the spring semester included coding an online numerical differentiator to determine vessel speed and acceleration from the position coordinates and vessel rotation rate from the orientation. This was done in the code's main loop by using a backwards difference method.

13.5 Large Scale Experimental Vessel

As the year progressed, the author constructed the control board of the vessel first. The control board included all of the vital electronics of the vessel, mainly the on-board computer

and motor amplifiers. After constructing the control board, the author and his advisors decided to increase the scale of the experimental vessel to an actual in-water large scale vessel consisting of a 10 foot boat using trolling motors as thruster pods as shown in Figure 22. Problems with the ordering process resulted in supplies arriving too late for implementation. Due to the order delays, the author and his advisors decided to pursue two experimental vessels, a small scale experimental vessel and a large scale on-water experimental vessel. The primary vessel for experimentation was the small scale vessel. The purpose of the large scale vessel was to provide an on-water demonstration platform. Data collection for determination of the tradeoff between performance and independence was provided by experimentation using the small scale vessel; therefore, this part of the experimentation aspect of the project was deferred to future work.

Figure 22: Large Scale Experimental Vessel

14. Conclusion

14.1 Contributions

This research has made several notable contributions; specifically, to the field of Control Systems Engineering and to ongoing research at the United States Naval Academy. The deliverables of this project are sorted using the previous categories as discussed in the following paragraphs.

14.1.1 Contributions to Control Systems Engineering

This research has made three major contributions to the state of the art of control systems engineering. These three contributions were the derivation, proof, simulation, and experimentation of Controller I; the derivation, proof, simulation, and experimentation of Controller III; and the performance verses independence analysis of Controllers I-III. Each one of these contributions will be detailed in the following paragraphs.

1. Although previous work has studied some aspects of Scenario I, Controller I is novel because it is the first control algorithm to employ unidirectional control inputs for all three degrees of freedom in the model. Essentially, Controller I is unique in its placement of the tugboats around the barge and its use of a commutation strategy to ensure that tugboats are only pushing against the hull. No previous work has addressed the problem of tugboat manipulation of a barge in this way. Controller I is superior in some aspects to the previous work because it allows control of the barge by only using positive force from each tugboat. This is desirable because pushing is generally more efficient in marine applications. Current marine propulsion systems are vastly more efficient when operation in the positive (pushing) direction as compared to the reverse

direction. Controller I takes advantage of this inherent efficiency in current marine propulsion.

2. The derivation and proof of Controller III in and of itself is a notable contribution to the field of control systems engineering. Controller III is the first control algorithm of its type, complexity, and application to be simulated and experimentally proven. Another substantial contribution is the identification of needed improvements on Controller III, as detailed in Section 14.2. This research essentially developed Controller III to solve Scenario III and then identified that future work is needed to have a controller that is completely independent on knowledge of the thrust configuration. An important discovery during the derivation process of Controller III was the requirement of some *a priori* knowledge about the signs of each element in the $\hat{B}$ matrix. This requirement was needed because Controller III uses the inverted $\hat{B}$ matrix to determine the proper thrust allocation to the tug boats. When inverting the $\hat{B}$ matrix, it is important that it maintains full rank. If the elements in the $\hat{B}$ matrix are left unbounded, they could potentially run through zero, causing the inverted matrix to loose full rank and the system to become uncontrollable, meaning that there is no possible solution set to drive the filtered tracking error to zero. Also, if the elements in the $\hat{B}$ matrix are left unbounded and they run through zero to values of the opposite sign it is also probable that the system will become unstable. This loss of stability is due to the fact that the lever arm for the torque terms manipulating the vessel's orientation changes signs. When this happens, the controller essentially believes it is applying a torque on the system to rotate in one direction, and in actuality it is rotating the opposite direction. This quickly causes the system to loose

stability and controllability. The required *a priori* knowledge of the signs of the $\hat{B}$ matrix motivated the need for the future work explained in Section 14.2.

3. The performance analysis detailed in Section 11, was also a significant contribution to the field of control systems engineering. The performance and independence analysis allowed this research to make several recommendations about the field ability of each controller. The performance analysis section discovered that the hydrodynamic properties used in Controller I were not accurate. This discovery is important because it is now apparent that an adaptive update law for drag and other hydrodynamic terms only helps the performance of a controller. Using adaptive control to account for inexact hydrodynamic drag measurement may be more viable than actually determining the drag due to the complex process required to determine its correct values. Future work, detailed in Section 14.2, will further investigate this discovery and make a recommendation as to whether or not adaptive update laws should be included in the controllers of each Scenario and Scenarios defined in the future.

14.1.2 Contributions to Ongoing Research

Many contributions to other research at the United States Naval Academy have been made by this project. Specifically, contributions such as the experimental vessel, Controller II, simulation infrastructure, and the developed vision system will help current and future research of this topic. These contributions will be detailed below.

1. Experimental vessel design and implementation will help future research on this problem and promote development of Scenarios and controllers by providing a reliable and easily reconfigurable platform for their experimentation. This system has already been designed and integrated in such a way to allow for growth. As future controllers are

developed, the hardware and code background already exists and is easily modified. This platform was designed to be rugged and to be easily reconfigurable.

2. Although previous work has already used adaptive controllers to account for unknown hydrodynamic drag, the development of Controller II was instrumental in performance analysis and will be incorporated in future work. More testing will be done, as detailed in Section 14.2, to future characterize the specific advantages inherent to Controller II's design. The possibility of implementing the adaptive update law developed in Controller II on other Scenarios will be investigated.
3. Much like the experimental vessel's contribution, the simulation infrastructure will contribute significantly to ongoing research. As detailed in Section 12, the simulation infrastructure was designed to be readily reconfigurable. This infrastructure requires only a few changes to successfully simulate a new controller.
4. The vision system developed in this research and described in Section 13.3 has also contributed significantly to ongoing research in this topic and others. This system has proven to be very reliable and invariant to changes in ambient light, and it can be and easily applied to other research. Although this vision system is very reliable, improvements in its design are possible and are detailed in Section 14.2.

14.2 Future Work

Although this Trident project investigated many problems associated with autonomous manipulation of a barge with robotic tugboats, there still remains many areas of the topic that require further research. Future work on the project will be listed and explained below in detail.

1. To integrate the disparities between simulation and experimentation, a hardware-in-the-loop (HIL) simulation needs to be developed. A HIL simulation uses mathematical

representations of many complex phenomena in order to provide more a realistic model. A HIL simulation also uses the actual control code and processor. For this project, the internal subsystems that need to be more accurately modeled include: tugboat thrust output, hydrodynamic drag (to include the effects of each tugboat and lateral movement), and the mass matrix (to include added mass effects). In short, to have the simulation more realistically model the experiment the hydrodynamic and physical properties of the vessel must be more closely modeled.

2. One area of future work that is needed to research item number one is a more complete understanding of the hydrodynamic effects of the experimental vessel. Early in the project, it was thought that, due to the system's slow speed, hydrodynamic effects would be negligible in a controlled environment. This was not true, because even in the very controlled environment of the Naval Academy's tow tank hydrodynamics still significantly affected the system. The single biggest hydrodynamic effect that needs to be modeled for a more accurate system is hydrodynamic drag. Computational fluid dynamics (CFD) could be used to more accurately model the highly non-linear drag effects experienced by a ship moving laterally. Once a CFD model is developed, the drag effects of each tugboat could be included rather than assuming that the tugboats have no effect on the system's hydrodynamic properties.
3. To help the debugging process, work needs to done to allow the MATLAB control files to run without input from the camera. Current control files require a video input to run, and this is not very conducive to testing changes in the code. Each control file needs to be changed so that the vision system can be turned off and positional data can be read in from a preexisting file. The current arrangement is fully operational, but is not ideal.

4. To test the reliability of the control algorithms with environmental disturbances, they need to be tested on an open-water experimental vessel. Currently, a large scale experimental vessel is under construction and is almost ready for a hardware-in-the-loop simulation. This vessel currently uses the same control code and basic setup, however, GPS integration is planned for the future. Additionally, this vessel will soon be tested in the tow tank to ensure the system has been properly integrated.
5. Hardware changes that should be implemented to improve the small scale experimental vessel include a wider field-of-view camera and replacing the Serial Expander Rabbit microcontroller with a PC104 microcomputer. Currently, the field-of-view of the webcam utilized is only large enough to do very limited experimentation. Quantifying performance was difficult because there was simply not enough space for the performance between controllers to be quantified. Although there are cameras with wider fields of view, they tend to have more distortion at the edges of the image. This could be countered by either placing the camera further above the water or by using a multi-camera system. Multi-camera systems use the images from multiple cameras to completely cover an area without distortion by carefully piecing the images together. Multi-camera systems are complicated and expensive but offer better coverage of the workspace. A camera system may be the only option to increase the size of the workspace in the tow tank because placing the camera higher above the water is not feasible. Changing the vessel's on-board computer is needed because current code has already exceeded the Rabbit microcontroller's memory. The solution used this year was to move the control code off of the vessel's on-board computer and on to the base station. Even though this solution gave the same results as the previous setup, it is ideal

to have the control code running on-board the vessel. The original setup allowed the control code to be processed at a faster rate than the video feed. This is desirable because although new positional data is only available at the video update rate (approximately 2 Hz), the control code could be run at a much faster rate by utilizing estimated data from an observer. Observers use data from a slow sensor to estimate the signal between updates. Running the control code at a faster rate is desirable because it greatly improves the response of the system.

6. Although Controller III offers a viable solution to the problem of unknown tugboat location, it still requires some knowledge of the system. As stated in 10.1, Controller III requires knowledge of the sign of each element in $\hat{B}$. This requirement is not ideal, as the information may not be available in real world applications. There are three solutions to this problem: Nassbaum Gains, root-searching functions, and a switching controller [17, 18]. Nassbaum Gains and root-searching functions continuously change the parameter's sign until they discover the correct value. Nassbaum gains are not robust and have not been successfully implemented [17]. Root-searching functions do not have the robustness problems of Nassbaum gains but have only been solved for simplified cases, nothing close to the complexity of this system [17]. Switching controllers start with a parameter identification phase and then move to the actual control phase. The parameter identification phase uses a special observer to determine the sign of unknown parameters and then uses an adaptive controller much like Controller III to determine the amplitude of the parameter [18]. Switching controllers offer the most viable solution; however, if the signs of the parameters are incorrect then there is not way to control the system. Future work will determine if there is a viable

closed-form control expression, and if there is not, will implement one of the above solutions.

15. **Bibliography**

[1] M. Feemster, J. Esposito, and J. Nicholson. "Manipulation of Large Objects by Swarms of Autonomous Marine Vehicles: Part I – Rotation." *Proceedings of the Southeastern Symposium on System Theory*, Cookville, TN, March 2006, Page(s) 205-209.

[2] H. G. Tanner, Ali Jadbabaie and George J. Pappas, "Stable Flocking of Mobile Agents, Part I: Fixed Topology," *42nd IEEE Conference on Decision and Control,* Maui Hawaii, December 2003, Page(s) 2010-2015.

[3] A. Jadbabaie, J. Lin and A. Morse, "Coordination of Groups of Mobile Autonomous Robots Using Nearest Neighbor Rules," *IEEE Transactions on Automatic Control*, Vol. 48, No. 6, 2003, Page(s) 998-1001.

[4] R. Olfati, "Flocking for Multi-Agent Dynamic Systems: Algorithms and Theory," *IEEE Transactions on Automatic Control*, Vol. 49, No. 2, June 2004, Page(s) 401-420.

[5] Y.C. Tan, B.E. Bishop ," Combining Classical and Behavior-Based Control for Swarms of Cooperating Vehicles", 2005. *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, April 2005, Page(s) 2499 – 2504.

[6] J.M. Esposito, and T.W.Dunbar, "Maintaining wireless connectivity constraints for swarms in the presence of obstacles", *IEEE Conference on Robotics and Automation*, Orlando, FL 2006, Page(s) 946-951.

[7] M. Lynch, "Locally Controllable Manipulation by Stable Pushing," *IEEE Transactions on Robotics and Automation*, Vol. 15, No. 2, 1999, Page(s) 318-327.

[8] D. Rus, (1997). "Coordinated manipulation of objects in a plane", *Algorithmica*, 19(1/2), Page(s) 129-147.

[9] D. Rus and B. Donald and J. Jennings. "Moving furniture with teams of autonomous robots", *In IEEE/RSJ IROS*, 1995, Page(s) 235-242.

[10] A. Sudsang and J.Ponce. "A New Approach to Motion Planning for Disc-Shaped Robots Manipulating a Polygonal Object in the Plane," *In IEEE International. Conference on Robotics and Automation*, Page(s) 1068-1075.

[11] G. Pereira, V. Kumar, and M. Campos, "Decentralized Algorithms for Multi-Robot Manipulation via Caging," *International Journal of Robotics Research,* 2004, Page(s) 8-11.

[12] P. Song, and V. Kumar, "A Potential Field Based Approach to Multi-Robot Manipulation," *IEEE International Conference on Robotics and Automation*, 2002, Page(s) 1217-1222.

[13] S. Sastry and M. Bodson, *Adaptive Control: Stability, Convergence, and Robustness*, Englewood Cliff, NJ, Prentice-Hall, 1989.

[14] T. Johansen, T. Fossen, and S. Berge, "Constrained Nonlinear Control Allocation With Singularity Avoidance Using Sequential Quadratic Programming," *IEEE Transactions On Control Systems Technology*, 2004, Page(s) 211-216.

[15] W. C. Webster and J. Sousa, "Optimum allocation for multiple thrusters," in *Proc. Int. Soc. Offshore and Polar Engineers Conf*. (ISOPE-99), Brest, France, 1999.

[16] T. I. Fossen, *Marine Control Systems*. Norway, Marine Cybernetics, 2002.

[17] R. Ortega and A. Astolfi, "Nonlinear PI Control of Uncertain Systems: An Alternative to Parameter Adaptation," *Proceedings of the 40th IEEE Conference on Decision and Control*, 2001, Page(s) 1749-1754.

[18] G. Bartolini and A. Ferrara, "A Switching Controller for Systems with Hard Uncertainties," *IEEE Transaction of Circuits and Systems-I: Fundamental Theory and Applications*, August 2003, Page(s) 984-990.

16. Enclosures

16.1 Controller I Code

16.1.1 Controller I Simulation Code

SwarmDynKin:

```
function [sys,x0,str,ts] = SwarmDynKin(t,x,u,flag)

switch flag,
  % ================
  % Initialization
  % ================
  case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;

  % ================
  % Derivatives
  % ================
  case 1,
    sys=mdlDerivatives(t,x,u);

  % ================
  % Update
  % ================
  case 2,
    sys=mdlUpdate(t,x,u);

  % ================
  % Outputs
  % ================
  case 3,
    sys=mdlOutputs(t,x,u);

  % =====================
  % GetTimeOfNextVarHit
  % =====================
  case 4,
    sys=mdlGetTimeOfNextVarHit(t,x,u);

  % ================
  % Terminate
  % ================
  case 9,
    sys=mdlTerminate(t,x,u);

  % ================
  % Unexpected flags
  % ================
  otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
```

```
%% ======================================================================
% mdlInitializeSizes
% Return the sizes, initial conditions, and sample times for the S-function
% ======================================================================

function [sys,x0,str,ts]=mdlInitializeSizes
% Call simsizes for a sizes structure, fill it in and convert it to a sizes
% array.

% Note that in this example, the values are hard coded.  This is not a
% recommended practice as the characteristics of the block are typically
% defined by the S-function parameters.

sizes = simsizes;

sizes.NumContStates  = 6;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 12;
sizes.NumInputs      = 0;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1;   % at least one sample time is needed

sys = simsizes(sizes);
%(74.8)*pi/180.0
x0  = [5.59 0.5 83.6*pi/180 0.0 0.0 0.0]';

% str is always an empty matrix
str = [];

% Initialize the array of sample times
ts  = [0 0];

% end mdlInitializeSizes

%% ======================================================================
% mdlDerivatives
% Return the derivatives for the continuous states.
% ======================================================================
function sys=mdlDerivatives(t,x,u)
% Exchange of variables
Px  = x(1,1);
Py  = x(2,1);
psi = x(3,1);

Vx   = x(4,1);
Vy   = x(5,1);
Vpsi = x(6,1);

V  = [Vx Vy Vpsi]';
P  = [Px Py psi]';

[dP,dV,U,e] = DynamicsControl(P,V);
sys         = [dP; dV];
```

```
% end mdlDerivatives

%
================================================================================
% mdlUpdate
% Handle discrete state updates, sample time hits, and major time step
% requirements.
%
================================================================================
function sys=mdlUpdate(t,x,u)
sys = [];

% end mdlUpdate

%%
================================================================================
% mdlOutputs
% Return the block outputs.
%
================================================================================
function sys=mdlOutputs(t,x,u)

% Exchange of variables
Px  = x(1,1);
Py  = x(2,1);
psi = x(3,1);

Vx  = x(4,1);
Vy  = x(5,1);
Vpsi = x(6,1);

V = [Vx Vy Vpsi]';
P = [Px Py psi]';

[dP,dV,Us,e] = DynamicsControl(P,V);
%
================================================================================
% Output Vector
%
================================================================================
sys = [P',Us',e'];

% end mdlOutputs

%%
================================================================================
% mdlGetTimeOfNextVarHit
% Return the time of the next hit for this block.  Note that the result is
% absolute time.  Note that this function is only used when you specify a
% variable discrete-time sample time [-2 0] in the sample time array in
% mdlInitializeSizes.mdl
%
================================================================================
function sys=mdlGetTimeOfNextVarHit(t,x,u)
```

```
sampleTime = 1;    % Example, set the next hit to be one second later.
sys = t + sampleTime;

% end mdlGetTimeOfNextVarHit

%%
================================================================================
% mdlTerminate
% Perform any end of simulation tasks.
%
================================================================================
function sys=mdlTerminate(t,x,u)
sys = [];

% end mdlTerminate
```

DynamicsControl:

```
function [dP,dV,U,e] = DynamicsControl(P,V)
% ========================================
% System Parameters and Matrix Definitions
% ========================================
% Tugboat locations
r1     = 0.6;
alpha1 = (180.0)*pi/180.0;
theta1 = (0.0)*pi/180.0;

r2     = 0.27;
alpha2 = (270.0)*pi/180.0;
theta2 = (44.72)*pi/180.0;

r3     = 0.19;
alpha3 = (270.0)*pi/180;
theta3 = (90.0)*pi/180.0;

r4     = r1;
alpha4 = (0.0)*pi/180.0;
theta4 = (180.0)*pi/180.0;

r5     = r2;
alpha5 = (90.0)*pi/180.0;
theta5 = (360.0)*pi/180.0-theta2;

r6     = r3;
alpha6 = (90.0)*pi/180.0;
theta6 = (270.0)*pi/180.0;

% Thrust matrix
B1 = [cos(alpha1) cos(alpha2) cos(alpha3) cos(alpha4) cos(alpha5)
cos(alpha6)];
B2 = [sin(alpha1) sin(alpha2) sin(alpha3) sin(alpha4) sin(alpha5)
sin(alpha6)];
B3 = [r1*sin(alpha1-theta1) r2*sin(alpha2-theta2) r3*sin(alpha3-theta3) ...
      r4*sin(alpha4-theta4) r5*sin(alpha5-theta5) r6*sin(alpha6-theta6)];
```

```
B = [B1;
     B2;
     B3];

% For the above configuration
Bs  = [-1      0           0;
       0       -1          -1;
       0  -r2*cos(theta2)   0 ];

% Mass matrix
m  = 15.5129; % (kg)
Iz = 1.5849;  % (kgm^2)

M = [m   0   0;
     0   m   0;
     0   0   Iz];

% Damping matrix
D = [.05 0 0;
     0 0.05 0;
     0 0 .15];

%% Control
Px  = P(1,1);
Py  = P(2,1);
psi = P(3,1);

% Rotation Matrix
R = [cos(psi) -sin(psi) 0;
     sin(psi)  cos(psi) 0;
        0          0    1];
Pdot  = R*V;

% Desired Trajectories
Pd     = [2 2 (90.0)*pi/180.0]';
PdDot  = [0.0 0.0 0.0]';
PdDDot = [0.0 0.0 0.0]';

% Control gains
gamma0 = 0.0;
%Kp     = 0.05;
% Kr     = 0.5;
% alpha  = 1.0;
% Kr = [7 0 0;
%       0 6.0 0;
%       0 0 3];
Kr = [2 0 0;
      0 5.0 0;
      0 0 4];
alpha = [.3 0 0;
           0 .3 0;
           0 0 .35];
```

```
% Error signals
e    = Pd-P;
eDot = PdDot-Pdot;
r    = eDot+alpha*e;

S   = [0  1 0;
       -1 0 0;
        0 0 0];

Us   =
inv(R*inv(M)*Bs)*(PdDDot+alpha*eDot+P(3,1)*S*Pdot+Kr*r+R*inv(M)*D*R'*Pdot);
% Us   = inv(R*inv(M)*Bs)*(Kp*e);

u14 = Us(1,1);
u25 = Us(2,1);
u36 = Us(3,1);

u1 = 0.5*(u14 +sqrt(u14^2+gamma0^2));
u4 = 0.5*(-u14+sqrt(u14^2+gamma0^2));

u2 = 0.5*(u25 +sqrt(u25^2+gamma0^2));
u5 = 0.5*(-u25+sqrt(u25^2+gamma0^2));

u3 = 0.5*(u36 +sqrt(u36^2+gamma0^2));
u6 = 0.5*(-u36+sqrt(u36^2+gamma0^2));

if (u1>2.4)
    u1 = 2.4;
end
if (u4>2.4)
    u4 = 2.4;
end
if (u2>1.6)
    u2 = 1.6;
end
if (u3>1.6)
    u3 = 1.6;
end
if (u5>1.6)
    u5 = 1.6;
end
if (u6>1.6)
    u6 = 1.6;
end


U  = [u1 u2 u3 u4 u5 u6]';

%% Kinematics
dP = R*V;

%% Swarm/Barge System Dynamics
dV = inv(M)*(-D*V+B*U);
```

```
return;
```

PlotAngle:

```
close all
plot(Px,Py,'-b')
axis([0 8 0 4.4]);
hold on
for i = 1:3:151
plot([Px(i)+.2*cos(psi(i)) Px(i)-.2*cos(psi(i))], [Py(i)+0.2*sin(psi(i))
Py(i)-0.2*sin(psi(i))],'-k');
plot(Px(i)+.2*cos(psi(i)),Py(i)+0.2*sin(psi(i)),'*g');
plot(Px(i)-.2*cos(psi(i)),Py(i)-0.2*sin(psi(i)),'*r')
end
title('Simulated Position and Orientation for Full Controller using IC3')
xlabel('X position (m)')
ylabel('Y position (m)')
```

16.1.2 Controller I Experimental Code

FullControl:

```
clear M
clear all
%desired location and orientation in pixels
x_des=input('Input the desired X position');
y_des=input('Input the desired Y position');
angle_des=input('Input the desired angle');
runtime = input('Input the run time');
%=========================================================================
%initialize variables and vessel parameters
%=========================================================================
x_log = []; y_log = []; si_log = []; U1_log = []; U2_log = [];
U3_log = []; U4_log = []; U5_log = []; U6_log = []; time_log = [];
x_conversion = 7.927/640;
y_conversion = 4.4/480;
Kr = [.2 0 0;
      0 .5 0;
      0 0 .4];
k_alpha = [.3 0 0;
           0 .3 0;
           0 0 .35];
% Kr = [.7 0 0;
%       0 .6 0;
%       0 0 .8];
% k_alpha = [.17 0 0;
%            0 .17 0;
%            0 0 .3];
a1 = 180*pi/180;
a2 = 270*pi/180;
a3 = 270*pi/180;
j = 1.5849;  %using moment of inertia formula for a cuboid from
```

```
%http://www.diracdelta.co.uk/science/source/m/o/moments%20of%20inertia/source
.html
m = 15.5129;
r1 = 0.6;
r2 = 0.27;
r3 = 0.19;
si = 0;
t1 = 0*pi/180;
t2 = 44.72*pi/180;
t3 = 90*pi/180;
xd = x_des;
yd = y_des;
sid = angle_des*pi/180;
gam0 = 0;
error_age = 0;
timer = 0;
time_age = 0;

s1 = [ 0  1 0;
      -1  0 0;
       0  0 0];

Mass = [m 0 0;
        0 m 0;
        0 0 j];

% B = [cos(a1) cos(a2) cos(a3);
%      sin(a1) sin(a2) sin(a3);
%      r1*(cos(t1)*sin(a1)-sin(t1)*cos(a1))...
%      r2*(cos(t2)*sin(a2)-sin(t2)*cos(a2))...
%      r3*(cos(t3)*sin(a3)-sin(t3)*cos(a3))];

B = [-1 0 0;
     0 -1 -1;
     0 -.192 0];

 Drag = [0.05 0 0;0 0.05 0;0 0 0.15];

tolerance = 0.5;
a_tol = 10;
flag = 0;
%=========================================================================
%set up com link
%=========================================================================

format compact
plotimage = 1;
communication = 1;
if (communication ==1)
    s3 = serial('COM1','baudrate',19200);
    fopen(s3)
end
xd_pix = xd*x_conversion^-1;
yd_pix = yd*y_conversion^-1;
%=========================================================================
```

```
%initialize camera
%=========================================================================
%  I think camera needs to be plugged in and creative cam software is off
before you start matlab
% set up video object
 vidobj = videoinput('winvideo');
% now you must trigger it to log data
triggerconfig(vidobj,'manual');
% only log a single frame
set(vidobj, 'framespertrigger', 1);
% lets you trigger it as many times as you want
set(vidobj, 'triggerrepeat', inf)
pose_hist = [];
%start but don't trigger (log)
start(vidobj)
% wait for warm up
pause(1)

frame_times = [];
i = 1;
tic;

while(flag==0)

    %=========================================================================
    %GETS VIDEO FEED AND DETERMINES THE VEHICLES POSITION AND ATTITUDE
    %=========================================================================
    %log a frame
    trigger(vidobj);
    % load it into memory with time stamp
    [frame time] = getdata(vidobj);
    % just keeping track of how many frames per sec we are getting
    frame_times = [frame_times; time];

    if(1==1)
    % define robots are a certain region of color space
    green = frame(:,:,1) >230 & frame(:,:,2)<160 & frame(:,:,3)<150;
%actually this is red

    red = frame(:,:,1) < 130 & frame(:,:,2)>220 & frame(:,:,3)>50;  %actually
this is green
    % looking for 8 connected comopnents
    R =bwlabel(red,8);
    Y =bwlabel(green,8);
    % using the labeled matrix will use Dunbar's optimized property finder
    % this is the built in version
    s = regionprops(R, 'centroid', 'area');
    t = regionprops(Y, 'centroid', 'area');
    %for red led's
    centroids_L = cat(1, s.Centroid);
    area_L = cat(1, s.Area);
    robots_L =  find(area_L>50);
    %for white led's
    centroids_R = cat(1, t.Centroid);
    area_R = cat(1, t.Area);
```

```
    robots_R =  find(area_R>50);
     % robots have to be bigger than some # of pixels to eliminate
     % spurrious results
     %can display to screen at cost of computation time...actually this
     %barely impacts it
    end
    if (plotimage == 1)
        if (plotimage == 1)
            imagesc(frame)
            figure(1)
            hold on
        end
     if ~isempty(robots_R)
     if ~isempty(robots_L)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%PLOTS THE POSITION AND ORIENTATION ON THE IMAGE%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        if (plotimage == 1)

            plot(centroids_R(robots_R,1), centroids_R(robots_R,2), 'k*');

            plot(xd_pix,480-yd_pix,'r+');

            plot([xd_pix+50*cos(sid) xd_pix-50*cos(sid)], [480-
(yd_pix+50*sin(sid)) 480-(yd_pix-50*sin(sid))]);

            plot(centroids_L(robots_L,1), centroids_L(robots_L,2), 'c*');

            plot([centroids_R(robots_R(1),1) centroids_L(robots_L(1),1)]...
                , [centroids_R(robots_R(1),2) centroids_L(robots_L(1),2)],
'k');

            plot(1/2*(centroids_R(robots_R(1),1)-
centroids_L(robots_L(1),1))+centroids_L(robots_L(1),1)...
                , 1/2*(centroids_R(robots_R(1),2)-
centroids_L(robots_L(1),2))+centroids_L(robots_L(1),2), 'r*');
        end
        if (i<runtime)
           M(i) = getframe;
        end
    %from green to red LED measured from normal x-axis on cartesian
    %coordinates
    si = atan2(centroids_R(robots_R(1),2)-(centroids_L(robots_L(1),2))...
        ,(centroids_L(robots_L(1),1))-centroids_R(robots_R(1),1))*180/pi;

    %makes sure angle is from 0 to 360 degrees
     if (si < 0)
         si = si + 360;
     end
     %converts to radians
     si = si*pi/180;

     error_si = sid-si;
```

```
     if (error_si > pi)
         error_si = error_si-(360*pi/180);
     end
     if (error_si < -pi)
            error_si = error_si+(360*pi/180);
     end


     pos = [1/2*(centroids_R(robots_R(1),1)-
centroids_L(robots_L(1),1))+centroids_L(robots_L(1),1)...
         , 1/2*(centroids_R(robots_R(1),2)-
centroids_L(robots_L(1),2))+centroids_L(robots_L(1),2)];
     x_pose = pos(1);
     y_pose = 480-pos(2);

     %Logs x,y,si,and time
     x_log(i)=x_pose;
     y_log(i)=y_pose;
     si_log(i) = si;
     timer = toc;
     time_log(i) = timer;

    end
    end




%++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
%++++++++++++++++++++++START CONTROL++++++++++++++++++++++++++++++++++++++++
%++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%DEFINE MEMBER MATRICIES%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Rot = [cos(si) -sin(si) 0 ;
       sin(si)  cos(si) 0 ;
          0         0   1];

%converts pixels  to m
x = x_pose*x_conversion;
y = y_pose*y_conversion;


% vectors
error = [xd-x yd-y error_si]';
Pd = [xd yd sid]';
P1 = [x y si]';
%=======================================
%CALCULATE ERROR DERIVATIVE
%=======================================
```

```
% derivatives using rise over run technique
if (i==1)
    error_dot = [0 0 0]';
    Pd_dot = [0 0 0]';
    P1_dot = [0 0 0]';
    Pd_dot_dot = [0 0 0]';
else
    error_dot = (error-error_age)/(timer-time_age);
    Pd_dot = (Pd-Pd_age)/(timer-time_age);
    P1_dot = (P1-P1_age)/(timer-time_age);
    Pd_dot_dot = (Pd_dot-Pd_dot_age)/(timer-time_age);
end
%error signal r
r_error = error_dot + k_alpha * error;

%//================================================
%//implement full controller
%...
%//================================================
U = inv(Rot*inv(Mass)*B)*((Pd_dot_dot+k_alpha*error_dot)+(Kr*r_error)...
+(P1_dot(3)*s1*P1_dot)+(Rot*inv(Mass)*Drag*Rot'*P1_dot));
%Allocate thrusts
Ua = U(1);
Ub = U(2);
Uc = U(3);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%AGE THE LOCATION TERMS%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Pd_age = Pd;
P1_age = P1;
Pd_dot_age = Pd_dot;
error_age = error;
time_age = timer;
%++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
%++++++++++++++++++++++++END CONTROL++++++++++++++++++++++++++++++++++++++++++++
%++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%SEND THRUSTS TO THE BOAT%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        if (communication ==1)

                my_string1 = sprintf('a%5.3f\r', Ua);
                fprintf(s3, my_string1);

                my_string2 = sprintf('b%5.3f\r', Ub);
                fprintf(s3, my_string2);

                my_string3 = sprintf('c%5.3f\r', Uc);
                fprintf(s3, my_string3);

        end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%CALCULATE AND LOG INDIVIDUAL THRUSTS%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

            U1 = 0.5*(Ua+sqrt(Ua*Ua+gam0*gam0));
            U4 = abs(0.5*(-Ua+sqrt(Ua*Ua+gam0*gam0)));

            U2 = 0.5*(Ub+sqrt(Ub*Ub+gam0*gam0));
            U5 = abs(0.5*(-Ub+sqrt(Ub*Ub+gam0*gam0)));

            U3 = 0.5*(Uc+sqrt(Uc*Uc+gam0*gam0));
            U6 = abs(0.5*(-Uc+sqrt(Uc*Uc+gam0*gam0)));

            if (U1>2.4)
                U1 = 2.4;
            end
            if (U4>2.4)
                U4 = 2.4;
            end
            if (U2>1.6)
                U2 = 1.6;
            end
            if (U3>1.6)
                U3 = 1.6;
            end
            if (U5>1.6)
                U5 = 1.6;
            end
            if (U6>1.6)
                U6 = 1.6;
            end

            U1_log(i) = U1;
            U4_log(i) = U4;
            U2_log(i) = U2;
            U5_log(i) = U5;
            U3_log(i) = U3;
            U6_log(i) = U6;


    end


%if ((x_des-tolerance) < x_pose && (x_des+tolerance) > x_pose && (y_des-
tolerance) < y_pose &&...
%       (y_des+tolerance) > y_pose && (angle_des - a_tol) < si*180/pi &&
(angle_des+a_tol)> si*180/pi)
if(i>runtime)
   flag = 1;
end
i=i+1


end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%RESET VARIABLES AND PROPERLY TERMINATE PROGRAM%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
flag = 0;
avg_time = mean(frame_times(3:end) - frame_times(2:end-1))
%close all
%plot(x_m_p*pose_hist(:,1), y_m_p*po se_hist(:,2),'*-')
%grid on;

% This is frame time
%plot(frame_times(3:end) - frame_times(2:end-1))


delete(vidobj);
if (communication ==1)
fclose(s3);
delete(s3);
end
```

16.2 Controller II Code

16.2.1 Controller II Simulation Code

SwarmDynKin:

```
function [sys,x0,str,ts] = SwarmDynKin(t,x,u,flag)

switch flag,
  % ================
  % Initialization
  % ================
  case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;

  % ================
  % Derivatives
  % ================
  case 1,
    sys=mdlDerivatives(t,x,u);

  % ================
  % Update
  % ================
  case 2,
    sys=mdlUpdate(t,x,u);

  % ================
  % Outputs
  % ================
  case 3,
    sys=mdlOutputs(t,x,u);

  % ====================
```

```
  % GetTimeOfNextVarHit
  % ====================
  case 4,
    sys=mdlGetTimeOfNextVarHit(t,x,u);

  % ================
  % Terminate
  % ================
  case 9,
    sys=mdlTerminate(t,x,u);

  % ================
  % Unexpected flags
  % ================
  otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end

%% ======================================================================
% mdlInitializeSizes
% Return the sizes, initial conditions, and sample times for the S-function
% ======================================================================

function [sys,x0,str,ts]=mdlInitializeSizes
% Call simsizes for a sizes structure, fill it in and convert it to a sizes
% array.

% Note that in this example, the values are hard coded.  This is not a
% recommended practice as the characteristics of the block are typically
% defined by the S-function parameters.

sizes = simsizes;

sizes.NumContStates  = 9;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 15;
sizes.NumInputs      = 0;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1;   % at least one sample time is needed

sys = simsizes(sizes);

x0  = [5.10 1.53 (285.0)*pi/180.0 0.0 0.0 0.0 0.05 0.05 0.15]';

% str is always an empty matrix
str = [];

% Initialize the array of sample times
ts  = [0 0];

% end mdlInitializeSizes

%% ======================================================================
% mdlDerivatives
```

```
% Return the derivatives for the continuous states.
% ================================================================================
function sys=mdlDerivatives(t,x,u)
% Exchange of variables
Px  = x(1,1);
Py  = x(2,1);
psi = x(3,1);

Vx   = x(4,1);
Vy   = x(5,1);
Vpsi = x(6,1);

D1 = x(7,1);
D2 = x(8,1);
D3 = x(9,1);


V  = [Vx Vy Vpsi]';
P  = [Px Py psi]';
theta_hat = [D1 D2 D3]';

[theta_hat_dot,dP,dV,U,e] = DynamicsControl(theta_hat,P,V);
sys         = [ dP; dV; theta_hat_dot];

% end mdlDerivatives

%
================================================================================
% mdlUpdate
% Handle discrete state updates, sample time hits, and major time step
% requirements.
%
================================================================================
function sys=mdlUpdate(t,x,u)
sys = [];

% end mdlUpdate

%%
================================================================================
% mdlOutputs
% Return the block outputs.
%
================================================================================
function sys=mdlOutputs(t,x,u)

% Exchange of variables
Px  = x(1,1);
Py  = x(2,1);
psi = x(3,1);

Vx  = x(4,1);
Vy  = x(5,1);
Vpsi = x(6,1);
```

```
D1 = x(7,1);
D2 = x(8,1);
D3 = x(9,1);

V = [Vx Vy Vpsi]';
P = [Px Py psi]';
theta_hat = [D1 D2 D3]';

[theta_hat_dot,dP,dV,Us,e] = DynamicsControl(theta_hat,P,V);
%
=============================================================================
% Output Vector
%
=============================================================================
sys = [P',Us',e',theta_hat'];

% end mdlOutputs

%%
=============================================================================
% mdlGetTimeOfNextVarHit
% Return the time of the next hit for this block.  Note that the result is
% absolute time.  Note that this function is only used when you specify a
% variable discrete-time sample time [-2 0] in the sample time array in
% mdlInitializeSizes.mdl
%
=============================================================================
function sys=mdlGetTimeOfNextVarHit(t,x,u)
sampleTime = 1;    % Example, set the next hit to be one second later.
sys = t + sampleTime;

% end mdlGetTimeOfNextVarHit

%%
=============================================================================
% mdlTerminate
% Perform any end of simulation tasks.
%
=============================================================================
function sys=mdlTerminate(t,x,u)
sys = [];

% end mdlTerminate
```

DynamicsControl:

```
function [theta_hat_dot,dP,dV,U,e] = DynamicsControl(theta_hat,P,V)
% =======================================
% System Parameters and Matrix Definitions
% =======================================
% Tugboat locations
r1     = 0.6;
alpha1 = (180.0)*pi/180.0;
```

```
theta1 = (0.0)*pi/180.0;

r2     = 0.27;
alpha2 = (270.0)*pi/180.0;
theta2 = (44.72)*pi/180.0;

r3     = 0.19;
alpha3 = (270.0)*pi/180;
theta3 = (90.0)*pi/180.0;

r4     = r1;
alpha4 = (0.0)*pi/180.0;
theta4 = (180.0)*pi/180.0;

r5     = r2;
alpha5 = (90.0)*pi/180.0;
theta5 = (360.0)*pi/180.0-theta2;

r6     = r3;
alpha6 = (90.0)*pi/180.0;
theta6 = (270.0)*pi/180.0;

d1 = 0.05;
d2 = 0.05;
d3 = 0.15;
gam1 = 1;
gam2 = 1;
gam3 = 1;


% Thrust matrix
B1 = [cos(alpha1) cos(alpha2) cos(alpha3) cos(alpha4) cos(alpha5)
cos(alpha6)];
B2 = [sin(alpha1) sin(alpha2) sin(alpha3) sin(alpha4) sin(alpha5)
sin(alpha6)];
B3 = [r1*sin(alpha1-theta1) r2*sin(alpha2-theta2) r3*sin(alpha3-theta3) ...
      r4*sin(alpha4-theta4) r5*sin(alpha5-theta5) r6*sin(alpha6-theta6)];

B = [B1;
     B2;
     B3];

% For the above configuration
Bs  = [-1      0           0;
       0      -1          -1;
       0  -r2*cos(theta2)   0 ];

% Mass matrix
m  = 15.5129; % (kg)
Iz = 1.5849;  % (kgm^2)

M = [m   0   0;
     0   m   0;
     0   0   Iz];
```

```
%% Control
Px  = P(1,1);
Py  = P(2,1);
psi = P(3,1);

% Rotation Matrix
R = [cos(psi) -sin(psi) 0;
     sin(psi)  cos(psi) 0;
        0          0    1];
Pdot  = R*V;

% Desired Trajectories
Pd     = [2 2 (90.0)*pi/180.0]';
PdDot  = [0.0 0.0 0.0]';
PdDDot = [0.0 0.0 0.0]';

% Control gains
gamma0 = 0.0;
%Kp     = 0.05;
% Kr     = 0.5;
% alpha  = 1.0;
% Kr = [.6 0 0;
%       0 .6 0;
%       0 0 .8];

Kr = [2.0 0 0;
      0 5.0 0;
      0 0 4.0];

alpha = [.3 0 0;
           0 .3 0;
           0 0 .35];

gamma = [gam1 0 0;
         0 gam2 0
         0 0 gam3];

D = [d1 0 0;
     0 d2 0;
     0 0 d3];
Y = [cos(psi)*Pdot(1)+sin(psi)*Pdot(2) 0 0;
     0 cos(psi)*Pdot(2)-sin(psi)*Pdot(1) 0
     0                0            Pdot(3)];

% Error signals
e    = Pd-P;
eDot = PdDot-Pdot;
r    = eDot+alpha*e;
theta_hat_dot = (r'*R*inv(M)*Y*gamma)';

S   = [0  1 0;
```

```
       -1 0 0;
        0 0 0];

Us   =
inv(R*inv(M)*Bs)*(PdDDot+alpha*eDot+P(3,1)*S*Pdot+Kr*r+R*inv(M)*Y*theta_hat);
% Us   = inv(R*inv(M)*Bs)*(Kp*e);

u14 = Us(1,1);
u25 = Us(2,1);
u36 = Us(3,1);

u1 = 0.5*(u14 +sqrt(u14^2+gamma0^2));
u4 = 0.5*(-u14+sqrt(u14^2+gamma0^2));

u2 = 0.5*(u25 +sqrt(u25^2+gamma0^2));
u5 = 0.5*(-u25+sqrt(u25^2+gamma0^2));

u3 = 0.5*(u36 +sqrt(u36^2+gamma0^2));
u6 = 0.5*(-u36+sqrt(u36^2+gamma0^2));

if (u1>2.4)
    u1 = 2.4;
end
if (u4>2.4)
    u4 = 2.4;
end
if (u2>1.6)
    u2 = 1.6;
end
if (u3>1.6)
    u3 = 1.6;
end
if (u5>1.6)
    u5 = 1.6;
end
if (u6>1.6)
    u6 = 1.6;
end


U  = [u1 u2 u3 u4 u5 u6]';

%% Kinematics
dP = R*V;

%% Swarm/Barge System Dynamics
dV = inv(M)*(-D*V+B*U);

return;
```

plot_angle:

```
close all
```

```
length = 151;
plot(Px,Py,'-b')
axis([0 8 0 4.4]);
hold on
for i = 1:3:length
plot([Px(i)+.2*cos(psi(i)) Px(i)-.2*cos(psi(i))], [Py(i)+0.2*sin(psi(i))
Py(i)-0.2*sin(psi(i))],'-k');
plot(Px(i)+.2*cos(psi(i)),Py(i)+0.2*sin(psi(i)),'*g');
plot(Px(i)-.2*cos(psi(i)),Py(i)-0.2*sin(psi(i)),'*r')
end
title('Simulated Position and Orientation for IC-1 using adaptive D
controller')
xlabel('X position (m)')
ylabel('Y position (m)')

figure(2)
subplot(2,2,1);
plot(Time,theta_hat(:,1),'-r')
title('Parameter:  D1')
xlabel('time (s)')
ylabel('Parameter Value')
subplot(2,2,2);
plot(Time,theta_hat(:,2),'-b')
title('Parameter:  D2')
xlabel('time (s)')
ylabel('Parameter Value')
subplot(2,2,3);
plot(Time,theta_hat(:,3),'-c')
title('Parameter:  D3')
xlabel('time (s)')
ylabel('Parameter Value')
```

16.2.2 Controller II Experimental Code

Adaptive_drag_exp:

```
clear M
clear all
%desired location and orientation in pixels
x_des=input('Input the desired X position');
y_des=input('Input the desired Y position');
angle_des=input('Input the desired angle');
runtime = input('Input the run time');
%=========================================================================
%initialize variables and vessel parameters
%=========================================================================
x_log = []; y_log = []; si_log = []; U1_log = []; U2_log = [];
U3_log = []; U4_log = []; U5_log = []; U6_log = []; time_log = [];

x_conversion = 7.927/640;
y_conversion = 4.4/480;
% Kr = [.2 0 0;
%       0 .5 0;
%       0 0 .6];
% k_alpha = [.3 0 0;
```

```
%            0 .3 0;
%            0 0 .35];
Kr = [.2 0 0;
      0 .5 0;
      0 0 .4];
k_alpha = [.3 0 0;
           0 .3 0;
           0 0 .35];
a1 = 180*pi/180;
a2 = 270*pi/180;
a3 = 270*pi/180;
j = 1.5849;  %using moment of inertia formula for a cuboid from
%http://www.diracdelta.co.uk/science/source/m/o/moments%20of%20inertia/source
.html
m = 15.5129;
r1 = 0.6;
r2 = 0.27;
r3 = 0.19;
si = 0;
t1 = 0*pi/180;
t2 = 44.72*pi/180;
t3 = 90*pi/180;
xd = x_des;
yd = y_des;
sid = angle_des*pi/180;
gam0 = 0;
error_age = 0;
timer = 0;
time_age = 0;

s1 = [ 0  1 0;
      -1  0 0;
       0  0 0];

Mass = [m 0 0;
        0 m 0;
        0 0 j];

B = [cos(a1) cos(a2) cos(a3);
     sin(a1) sin(a2) sin(a3);
     r1*(cos(t1)*sin(a1)-sin(t1)*cos(a1))...
     r2*(cos(t2)*sin(a2)-sin(t2)*cos(a2))...
     r3*(cos(t3)*sin(a3)-sin(t3)*cos(a3))];

d1 = 0.1;
d2 = 0.1;
d3 = 0.1;


%theta = [d1 d2 d3]';

gamma = [1 0 0
         0 1 0
         0 0 1];
```

```
flag = 0;
%=========================================================================
%set up com link
%=========================================================================

format compact
plotimage = 1;
communication = 1;
if (communication ==1)
    s3 = serial('COM1','baudrate',19200);
    fopen(s3)
end
xd_pix = xd*x_conversion^-1;
yd_pix = yd*y_conversion^-1;
%=========================================================================
%initialize camera
%=========================================================================
%  I think camera needs to be plugged in and creative cam software is off
before you start matlab
% set up video object
 vidobj = videoinput('winvideo');
% now you must trigger it to log data
triggerconfig(vidobj,'manual');
% only log a single frame
set(vidobj, 'framespertrigger', 1);
% lets you trigger it as many times as you want
set(vidobj, 'triggerrepeat', inf)
pose_hist = [];
%start but don't trigger (log)
start(vidobj)
% wait for warm up
pause(1)

frame_times = [];
i = 1;
tic;

while(flag==0)

    %=========================================================================
    %GETS VIDEO FEED AND DETERMINES THE VEHICLES POSITION AND ATTITUDE
    %=========================================================================
    %log a frame
    trigger(vidobj);
    % load it into memory with time stamp
    [frame time] = getdata(vidobj);
    % just keeping track of how many frames per sec we are getting
    frame_times = [frame_times; time];

    if(1==1)
    % define robots are a certain region of color space
    green = frame(:,:,1) >230 & frame(:,:,2)<160 & frame(:,:,3)<150;
%actually this is red
```

```
    red = frame(:,:,1) < 130 & frame(:,:,2)>220 & frame(:,:,3)>50;  %actually
this is green
    % looking for 8 connected comopnents
    R =bwlabel(red,8);
    Y =bwlabel(green,8);
    % using the labeled matrix will use Dunbar's optimized property finder
    % this is the built in version
    s = regionprops(R, 'centroid', 'area');
    t = regionprops(Y, 'centroid', 'area');
    %for red led's
    centroids_L = cat(1, s.Centroid);
    area_L = cat(1, s.Area);
    robots_L =  find(area_L>50);
    %for white led's
    centroids_R = cat(1, t.Centroid);
    area_R = cat(1, t.Area);
    robots_R =  find(area_R>50);
     % robots have to be bigger than some # of pixels to eliminate
     % spurrious results
     %can display to screen at cost of computation time...actually this
     %barely impacts it
    end
    if (plotimage == 1)
        if (plotimage == 1)
            imagesc(frame)
            figure(1)
            hold on
        end
     if ~isempty(robots_R)
     if ~isempty(robots_L)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%PLOTS THE POSITION AND ORIENTATION ON THE IMAGE%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        if (plotimage == 1)

            plot(centroids_R(robots_R,1), centroids_R(robots_R,2), 'k*');

            plot(xd_pix,480-yd_pix,'r+');

            plot([xd_pix+50*cos(sid) xd_pix-50*cos(sid)], [480-
(yd_pix+50*sin(sid)) 480-(yd_pix-50*sin(sid))]);

            plot(centroids_L(robots_L,1), centroids_L(robots_L,2), 'c*');

            plot([centroids_R(robots_R(1),1) centroids_L(robots_L(1),1)]...
                , [centroids_R(robots_R(1),2) centroids_L(robots_L(1),2)],
'k');

            plot(1/2*(centroids_R(robots_R(1),1)-
centroids_L(robots_L(1),1))+centroids_L(robots_L(1),1)...
                , 1/2*(centroids_R(robots_R(1),2)-
centroids_L(robots_L(1),2))+centroids_L(robots_L(1),2), 'r*');
        end
        if (i<runtime)
```

```
            M(i) = getframe;
        end
    %from green to red LED measured from normal x-axis on cartesian
    %coordinates
    si = atan2(centroids_R(robots_R(1),2)-(centroids_L(robots_L(1),2))...
        ,(centroids_L(robots_L(1),1))-centroids_R(robots_R(1),1))*180/pi;

    %makes sure angle is from 0 to 360 degrees
     if (si < 0)
         si = si + 360;
     end
     %converts to radians
     si = si*pi/180;

     error_si = sid-si;
     if (error_si > pi)
         error_si = error_si-(360*pi/180);
     end
     if (error_si < -pi)
             error_si = error_si+(360*pi/180);
     end


     pos = [1/2*(centroids_R(robots_R(1),1)-
centroids_L(robots_L(1),1))+centroids_L(robots_L(1),1)...
         , 1/2*(centroids_R(robots_R(1),2)-
centroids_L(robots_L(1),2))+centroids_L(robots_L(1),2)];
     x_pose = pos(1);
     y_pose = 480-pos(2);

     %Logs x,y,si,and time
     x_log(i)=x_pose;
     y_log(i)=y_pose;
     si_log(i) = si;
     timer = toc;
     time_log(i) = timer;

    end
    end
```

```
%++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
%++++++++++++++++++++++START CONTROL++++++++++++++++++++++++++++++++++++++++
%++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%DEFINE MEMBER MATRICIES%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Rot = [cos(si) -sin(si) 0 ;
       sin(si)  cos(si) 0 ;
```

```
          0         0   1];

%converts pixels  to m
x = x_pose*x_conversion;
y = y_pose*y_conversion;


% vectors
error = [xd-x yd-y error_si]';
Pd = [xd yd sid]';
P1 = [x y si]';
%=======================================
%CALCULATE ERROR DERIVATIVE
%=======================================

% derivatives using rise over run technique
if (i==1)
    error_dot = [0 0 0]';
    Pd_dot = [0 0 0]';
    P1_dot = [0 0 0]';
    Pd_dot_dot = [0 0 0]';
else
    error_dot = (error-error_age)/(timer-time_age);
    Pd_dot = (Pd-Pd_age)/(timer-time_age);
    P1_dot = (P1-P1_age)/(timer-time_age);
    Pd_dot_dot = (Pd_dot-Pd_dot_age)/(timer-time_age);
end

Y_ad = [cos(si)*P1_dot(1)+sin(si)*P1_dot(2) 0 0;
     0 cos(si)*P1_dot(2)-sin(si)*P1_dot(1) 0
     0                0            P1_dot(3)];

%error signal r
r_error = error_dot + k_alpha * error;
%%%%%%%need to set up integrator
theta_hat_dot = (r_error'*Rot*inv(Mass)*Y_ad*gamma)';

if(i==1)
theta_hat = [d1 d2 d3]';
else
theta_hat = theta_hat+((timer-time_age)*(theta_hat_dot +
theta_hat_dot_age)/2);
end

theta_hat_log(1,i) = theta_hat(1);
theta_hat_log(2,i) = theta_hat(2);
theta_hat_log(3,i) = theta_hat(3);

%//================================================
%//implement full controller
%...
%//================================================

 U = inv(Rot*inv(Mass)*B)*((Pd_dot_dot+k_alpha*error_dot)...
```

```
 +(P1_dot(3)*s1*P1_dot)+(Kr*r_error)+(Rot*inv(Mass)*Y_ad*theta_hat));
%U = inv(Rot*inv(Mass)*B)*((Kr*error));
%Allocate thrusts
Ua = U(1);
Ub = U(2);
Uc = U(3);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%AGE THE LOCATION TERMS%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Pd_age = Pd;
P1_age = P1;
Pd_dot_age = Pd_dot;
error_age = error;
time_age = timer;
theta_hat_dot_age = theta_hat_dot;
%++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
%++++++++++++++++++++++END CONTROL++++++++++++++++++++++++++++++++++++++++++++
%++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%SEND THRUSTS TO THE BOAT%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        if (communication ==1)

              my_string1 = sprintf('a%5.3f\r', Ua);
              fprintf(s3, my_string1);

              my_string2 = sprintf('b%5.3f\r', Ub);
              fprintf(s3, my_string2);

              my_string3 = sprintf('c%5.3f\r', Uc);
              fprintf(s3, my_string3);

        end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%CALCULATE AND LOG INDIVIDUAL THRUSTS%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

            U1 = 0.5*(Ua+sqrt(Ua*Ua+gam0*gam0));
            U4 = abs(0.5*(-Ua+sqrt(Ua*Ua+gam0*gam0)));

            U2 = 0.5*(Ub+sqrt(Ub*Ub+gam0*gam0));
            U5 = abs(0.5*(-Ub+sqrt(Ub*Ub+gam0*gam0)));

            U3 = 0.5*(Uc+sqrt(Uc*Uc+gam0*gam0));
            U6 = abs(0.5*(-Uc+sqrt(Uc*Uc+gam0*gam0)));

            if (U1>2.4)
                U1 = 2.4;
            end
            if (U4>2.4)
                U4 = 2.4;
            end
```

```
            if (U2>1.6)
                U2 = 1.6;
            end
            if (U3>1.6)
                U3 = 1.6;
            end
            if (U5>1.6)
                U5 = 1.6;
            end
            if (U6>1.6)
                U6 = 1.6;
            end

            U1_log(i) = U1;
            U4_log(i) = U4;
            U2_log(i) = U2;
            U5_log(i) = U5;
            U3_log(i) = U3;
            U6_log(i) = U6;


    end



if(i>runtime)
   flag = 1;
end
i=i+1


end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%RESET VARIABLES AND PROPERLY TERMINATE PROGRAM%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
flag = 0;
avg_time = mean(frame_times(3:end) - frame_times(2:end-1))
%close all
%plot(x_m_p*pose_hist(:,1), y_m_p*po se_hist(:,2),'*-')
%grid on;

% This is frame time
%plot(frame_times(3:end) - frame_times(2:end-1))


delete(vidobj);
if (communication ==1)
fclose(s3);
delete(s3);
end
```

16.3 Controller III Code

16.3.1 Controller III Simulation Code

Simulate:

```
global u14_age
global u25_age
global u36_age
u14_age = 0.0;
u25_age = 0.0;
u36_age = 0.0;
LogFreq = 1;
sim('SwarmModel',[0:0.25:200]);
ParseData
plot_angle
```

SwarmDynKin:

```
function [sys,x0,str,ts] = SwarmDynKin(t,x,u,flag)

switch flag,
  % ================
  % Initialization
  % ================
  case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;

  % ================
  % Derivatives
  % ================
  case 1,
    sys=mdlDerivatives(t,x,u);

  % ================
  % Update
  % ================
  case 2,
    sys=mdlUpdate(t,x,u);

  % ================
  % Outputs
  % ================
  case 3,
    sys=mdlOutputs(t,x,u);

  % =====================
  % GetTimeOfNextVarHit
  % =====================
  case 4,
    sys=mdlGetTimeOfNextVarHit(t,x,u);
```

```
  % ================
  % Terminate
  % ================
  case 9,
    sys=mdlTerminate(t,x,u);

  % ================
  % Unexpected flags
  % ================
  otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end

%% ==============================================================================
% mdlInitializeSizes
% Return the sizes, initial conditions, and sample times for the S-function
% ==============================================================================

function [sys,x0,str,ts]=mdlInitializeSizes
% Call simsizes for a sizes structure, fill it in and convert it to a sizes
% array.

% Note that in this example, the values are hard coded.  This is not a
% recommended practice as the characteristics of the block are typically
% defined by the S-function parameters.

sizes = simsizes;

sizes.NumContStates  = 15;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 33;
sizes.NumInputs      = 0;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1;   % at least one sample time is needed

sys = simsizes(sizes);

theta_hat_IC = [1.0 0 0 0 1 1 0 0.19 0];

x0  = [5.59 0.5 83.6*pi/180 0.0 0.0 0.0 theta_hat_IC]';

% str is always an empty matrix
str = [];

% Initialize the array of sample times
ts  = [0 0];

% end mdlInitializeSizes

%% ==============================================================================
% mdlDerivatives
% Return the derivatives for the continuous states.
% ==============================================================================
function sys=mdlDerivatives(t,x,u)
```

```
% Exchange of variables
Px  = x(1,1);
Py  = x(2,1);
psi = x(3,1);

Vx   = x(4,1);
Vy   = x(5,1);
Vpsi = x(6,1);

B1 = x(7,1);
B2 = x(8,1);
B3 = x(9,1);
B4 = x(10,1);
B5 = x(11,1);
B6 = x(12,1);
B7 = x(13,1);
B8 = x(14,1);
B9 = x(15,1);

V  = [Vx Vy Vpsi]';
P  = [Px Py psi]';
theta_hat = [B1 B2 B3 B4 B5 B6 B7 B8 B9]';

theta_hat_min = 0.1;

if(theta_hat(1)<= theta_hat_min)
    theta_hat(1) = theta_hat_min;
end

if(theta_hat(5)<= theta_hat_min)
    theta_hat(5) = theta_hat_min;
end

if(theta_hat(6)<= theta_hat_min)
    theta_hat(6) = theta_hat_min;
end

if(theta_hat(8)<= theta_hat_min)
    theta_hat(8) = theta_hat_min;
end

[theta_hat_dot,dP,dV,U,e,r,B_hat] = DynamicsControl(theta_hat,P,V);
sys         = [ dP; dV; theta_hat_dot];

% end mdlDerivatives

%
%=============================================================================
% mdlUpdate
% Handle discrete state updates, sample time hits, and major time step
% requirements.
%
%=============================================================================
function sys=mdlUpdate(t,x,u)
```

```
sys = [];

% end mdlUpdate

%%
================================================================================
% mdlOutputs
% Return the block outputs.
%
================================================================================
function sys=mdlOutputs(t,x,u)

% Exchange of variables
Px  = x(1,1);
Py  = x(2,1);
psi = x(3,1);

Vx   = x(4,1);
Vy   = x(5,1);
Vpsi = x(6,1);

B1 = x(7,1);
B2 = x(8,1);
B3 = x(9,1);
B4 = x(10,1);
B5 = x(11,1);
B6 = x(12,1);
B7 = x(13,1);
B8 = x(14,1);
B9 = x(15,1);

V  = [Vx Vy Vpsi]';
P  = [Px Py psi]';
theta_hat = [B1 B2 B3 B4 B5 B6 B7 B8 B9]';

theta_hat_min = 0.1;

if(theta_hat(1)<= theta_hat_min)
    theta_hat(1) = theta_hat_min;
end

if(theta_hat(5)<= theta_hat_min)
    theta_hat(5) = theta_hat_min;
end

if(theta_hat(6)<= theta_hat_min)
    theta_hat(6) = theta_hat_min;
end

if(theta_hat(8)<= theta_hat_min)
    theta_hat(8) = theta_hat_min;
end

[theta_hat_dot,dP,dV,U,e,r,B_hat] = DynamicsControl(theta_hat,P,V);
```

```
%
=============================================================================
% Output Vector
%
=============================================================================
sys = [P',U',e',theta_hat',r',B_hat(1,:),B_hat(2,:),B_hat(3,:)];

% end mdlOutputs

%%
=============================================================================
% mdlGetTimeOfNextVarHit
% Return the time of the next hit for this block.  Note that the result is
% absolute time.  Note that this function is only used when you specify a
% variable discrete-time sample time [-2 0] in the sample time array in
% mdlInitializeSizes.mdl
%
=============================================================================
function sys=mdlGetTimeOfNextVarHit(t,x,u)
sampleTime = 1;    % Example, set the next hit to be one second later.
sys = t + sampleTime;

% end mdlGetTimeOfNextVarHit

%%
=============================================================================
% mdlTerminate
% Perform any end of simulation tasks.
%
=============================================================================
function sys=mdlTerminate(t,x,u)
sys = [];

% end mdlTerminate
```

DynamicsControl:

```
function [theta_hat_dot,dP,dV,U,e,r,B_hat] = DynamicsControl(theta_hat,P,V)
% ========================================
% System Parameters and Matrix Definitions
% ========================================
% Tugboat locations
global u14_age
global u25_age
global u36_age

r1     = 0.6;
alpha1 = (180.0)*pi/180.0;
theta1 = (0.0)*pi/180.0;

r2     = 0.27;
alpha2 = (270.0)*pi/180.0;
theta2 = (44.72)*pi/180.0;
```

```
r3     = 0.19;
alpha3 = (270.0)*pi/180;
theta3 = (90.0)*pi/180.0;

r4     = r1;
alpha4 = (0.0)*pi/180.0;
theta4 = (180.0)*pi/180.0;

r5     = r2;
alpha5 = (90.0)*pi/180.0;
theta5 = (360.0)*pi/180.0-theta2;

r6     = r3;
alpha6 = (90.0)*pi/180.0;
theta6 = (270.0)*pi/180.0;

%B member signs needed for adaptive laws to work%%%%%%%
b11_sgn = -1;
b12_sgn = 0;
b13_sgn = 0;
b21_sgn = 0;
b22_sgn = -1;
b23_sgn = -1;
b31_sgn = 0;
b32_sgn = -1;
b33_sgn = 0;

d1 = 0.05;
d2 = 0.05;
d3 = 0.15;


D = [d1 0 0;0 d2 0;0 0 d3];


% Mass matrix
m  = 15.5129; % (kg)
Iz = 1.5849;  % (kgm^2)

M = [m   0   0;
     0   m   0;
     0   0   Iz];



%% Control
Px  = P(1,1);
Py  = P(2,1);
psi = P(3,1);

% Rotation Matrix
R = [cos(psi) -sin(psi) 0;
```

```
     sin(psi)  cos(psi) 0;
        0          0    1];
Pdot  = R*V;

% Desired Trajectories
Pd     = [2 2 (90.0)*pi/180.0]';
PdDot  = [0.0 0.0 0.0]';
PdDDot = [0.0 0.0 0.0]';

% Control gains
gamma0 = 0.0;
%Kp     = 0.05;
% Kr     = 0.5;
% alpha  = 1.0;
% Kr = [.6 0 0;
%       0 .6 0;
%       0 0 .8];

Kr = [2 0 0;
      0 5 0;
      0 0 4];

alpha = [.3 0 0;
           0 .3 0;
           0 0 .35];




%theta = [-1 0 0 0 -1 -1 0 -r2*cos(theta2) 0]';
B = [-1 0 0;
     0 -1 -1;
     0 -r2*cos(theta2) 0];
% theta =  [(B(1,1)) (B(1,2)) (B(1,3))
%           (B(2,1)) (B(2,2)) (B(2,3))
%           (B(3,1)) (B(3,2)) (B(3,3))]';


%%update Y
theta_hat_min = 0.1;

if(theta_hat(1)<= theta_hat_min)
    theta_hat(1) = theta_hat_min;
end

if(theta_hat(5)<= theta_hat_min)
    theta_hat(5) = theta_hat_min;
end

if(theta_hat(6)<= theta_hat_min)
    theta_hat(6) = theta_hat_min;
end
```

```
if(theta_hat(8)<= theta_hat_min)
    theta_hat(8) = theta_hat_min;
end

B_hat = [b11_sgn*(theta_hat(1)) b12_sgn*(theta_hat(2))
b13_sgn*(theta_hat(3));
         b21_sgn*(theta_hat(4)) b22_sgn*(theta_hat(5))
b23_sgn*(theta_hat(6));
         b31_sgn*(theta_hat(7)) b32_sgn*(theta_hat(8))
b33_sgn*(theta_hat(9))];



Y = [b11_sgn*(cos(psi)*u14_age)/m, b12_sgn*(cos(psi)*u25_age)/m,
b13_sgn*(cos(psi)*u36_age)/m,...
    b21_sgn*(sin(psi)*u14_age)/m, b22_sgn*(sin(psi)*u25_age)/m,
b23_sgn*(sin(psi)*u36_age)/m, 0, 0, 0;
    b11_sgn*-(sin(psi)*u14_age)/m, b12_sgn*-(sin(psi)*u25_age)/m, b13_sgn*-
(sin(psi)*u36_age)/m,...
    b21_sgn*(cos(psi)*u14_age)/m, b22_sgn*(cos(psi)*u25_age)/m,
b23_sgn*(cos(psi)*u36_age)/m, 0, 0, 0;
    0, 0, 0, 0, 0, 0, b31_sgn*u14_age/Iz, b32_sgn*u25_age/Iz,
b33_sgn*u36_age/Iz];

% Error signals
e    = Pd-P;
eDot = PdDot-Pdot;
r    = eDot+alpha*e;


theta_hat_dot = -1*Y'*r;



if (theta_hat(1) <= theta_hat_min && theta_hat_dot(1)<0.0)
    theta_hat_dot(1) = 0.0;
end

if (theta_hat(5) <= theta_hat_min && theta_hat_dot(5)<0.0)
    theta_hat_dot(5) = 0.0;
end

if (theta_hat(6) <= theta_hat_min && theta_hat_dot(6)<0.0)
    theta_hat_dot(6) = 0.0;
end

if (theta_hat(8) <= theta_hat_min && theta_hat_dot(8)<0.0)
    theta_hat_dot(8) = 0.0;
end
```

```
S   = [0  1 0;
       -1 0 0;
        0 0 0];

Us   =
inv(R*inv(M)*B_hat)*(PdDDot+alpha*eDot+P(3,1)*S*Pdot+Kr*r+R*inv(M)*D*R'*Pdot)
;


u14 = Us(1,1);
u25 = Us(2,1);
u36 = Us(3,1);

u14_age = u14;
u25_age = u25;
u36_age = u36;

u1 = 0.5*(u14 +sqrt(u14^2+gamma0^2));
u4 = 0.5*(-u14+sqrt(u14^2+gamma0^2));

u2 = 0.5*(u25 +sqrt(u25^2+gamma0^2));
u5 = 0.5*(-u25+sqrt(u25^2+gamma0^2));

u3 = 0.5*(u36 +sqrt(u36^2+gamma0^2));
u6 = 0.5*(-u36+sqrt(u36^2+gamma0^2));

if (u1>2.4)
    u1 = 2.4;
end
if (u4>2.4)
    u4 = 2.4;
end
if (u2>1.6)
    u2 = 1.6;
end
if (u3>1.6)
    u3 = 1.6;
end
if (u5>1.6)
    u5 = 1.6;
end
if (u6>1.6)
    u6 = 1.6;
end


U  = [u1 u2 u3 u4 u5 u6]';

%% Kinematics
dP = R*V;

%% Swarm/Barge System Dynamics
%dV = inv(M)*(-D*V+M*R'*Y*theta);
dV = inv(M)*(-D*V+B*Us);
```

```
return;
```

16.3.2 Controller III Experimental Code

Adaptive_B_exp:

```
clear M
clear all
close all
clc
%desired location and orientation in pixels
x_des=input('Input the desired X position');
y_des=input('Input the desired Y position');
angle_des=input('Input the desired angle');
frame_number = input('Input the run time');
%=========================================================================
%initialize variables and vessel parameters
%=========================================================================
x_log = []; y_log = []; si_log = []; U1_log = []; U2_log = [];
U3_log = []; U4_log = []; U5_log = []; U6_log = []; time_log = [];
B_hat_log = [];
x_conversion = 7.927/640;
y_conversion = 4.4/480;

theta_hat_min = 0.05;


% Kr = [.7 0 0;
%       0 .6 0;
%       0 0 .3];
% k_alpha = [.17 0 0;
%            0 .17 0;
%            0 0 .3];
       Kr = [.2 0 0;
             0 .5 0;
            0 0 .4];
       k_alpha = [.3 0 0;
                  0 .3 0;
                   0 0 .35];
% a1 = 180*pi/180;
% a2 = 270*pi/180;
% a3 = 270*pi/180;

%B member signs needed for adaptive laws to work%%%%%%%
b11_sgn = -1;
b12_sgn = 0;
b13_sgn = 0;
b21_sgn = 0;
b22_sgn = -1;
b23_sgn = -1;
b31_sgn = 0;
b32_sgn = -1;
b33_sgn = 0;
```

```
j = 1.5849;  %using moment of inertia formula for a cuboid from
%http://www.diracdelta.co.uk/science/source/m/o/moments%20of%20inertia/source
.html
m = 15.5129;
% r1 = 0.6;
% r2 = 0.27;
% r3 = 0.19;
si = 0;
% t1 = 0*pi/180;
% t2 = 44.72*pi/180;
% t3 = 90*pi/180;
xd = x_des;
yd = y_des;
sid = angle_des*pi/180;
gam0 = 0;
error_age = 0;
timer = 0;
time_age = 0;

s1 = [ 0  1 0;
      -1  0 0;
       0  0 0];

Mass = [m 0 0;
        0 m 0;
        0 0 j];

d1 = 0.05;
d2 = 0.05;
d3 = 0.15;


Drag = [d1 0 0;
        0 d2 0;
        0 0 d3];

Ua_age = 0.0;
Ub_age = 0.0;
Uc_age = 0.0;

% gamma = [1 0 0
%          0 1 0
%          0 0 1];

flag = 0;
%=====================================================================
%set up com link
%=====================================================================

format compact
plotimage = 1;
communication = 1;
if (communication ==1)
```

```
    s3 = serial('COM1','baudrate',19200);
    fopen(s3)
end
xd_pix = xd*x_conversion^-1;
yd_pix = yd*y_conversion^-1;
%=========================================================================
%initialize camera
%=========================================================================
%  I think camera needs to be plugged in and creative cam software is off
before you start matlab
% set up video object
 vidobj = videoinput('winvideo');
% now you must trigger it to log data
triggerconfig(vidobj,'manual');
% only log a single frame
set(vidobj, 'framespertrigger', 1);
% lets you trigger it as many times as you want
set(vidobj, 'triggerrepeat', inf)
pose_hist = [];
%start but don't trigger (log)
start(vidobj)
% wait for warm up
pause(1)

frame_times = [];
i = 1;
tic;

while(flag==0)

    %=========================================================================
    %GETS VIDEO FEED AND DETERMINES THE VEHICLES POSITION AND ATTITUDE
    %=========================================================================
    %log a frame
    trigger(vidobj);
    % load it into memory with time stamp
    [frame time] = getdata(vidobj);
    % just keeping track of how many frames per sec we are getting
    frame_times = [frame_times; time];

    if(1==1)
    % define robots are a certain region of color space
    green = frame(:,:,1) >230 & frame(:,:,2)<160 & frame(:,:,3)<150;
%actually this is red

    red = frame(:,:,1) < 130 & frame(:,:,2)>220 & frame(:,:,3)>50;  %actually
this is green
    % looking for 8 connected comopnents
    R =bwlabel(red,8);
    Y =bwlabel(green,8);
    % using the labeled matrix will use Dunbar's optimized property finder
    % this is the built in version
    s = regionprops(R, 'centroid', 'area');
    t = regionprops(Y, 'centroid', 'area');
    %for red led's
```

```
    centroids_L = cat(1, s.Centroid);
    area_L = cat(1, s.Area);
    robots_L =  find(area_L>50);
    %for white led's
    centroids_R = cat(1, t.Centroid);
    area_R = cat(1, t.Area);
    robots_R =  find(area_R>50);
     % robots have to be bigger than some # of pixels to eliminate
     % spurrious results
     %can display to screen at cost of computation time...actually this
     %barely impacts it
    end
    if (plotimage == 1)
        if (plotimage == 1)
            imagesc(frame)
            figure(1)
            hold on
        end
     if ~isempty(robots_R)
     if ~isempty(robots_L)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%PLOTS THE POSITION AND ORIENTATION ON THE IMAGE%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        if (plotimage == 1)

            plot(centroids_R(robots_R,1), centroids_R(robots_R,2), 'k*');

            plot(xd_pix,480-yd_pix,'r+');

            plot([xd_pix+50*cos(sid) xd_pix-50*cos(sid)], [480-
(yd_pix+50*sin(sid)) 480-(yd_pix-50*sin(sid))]);

            plot(centroids_L(robots_L,1), centroids_L(robots_L,2), 'c*');

            plot([centroids_R(robots_R(1),1) centroids_L(robots_L(1),1)]...
                , [centroids_R(robots_R(1),2) centroids_L(robots_L(1),2)],
'k');

            plot(1/2*(centroids_R(robots_R(1),1)-
centroids_L(robots_L(1),1))+centroids_L(robots_L(1),1)...
                , 1/2*(centroids_R(robots_R(1),2)-
centroids_L(robots_L(1),2))+centroids_L(robots_L(1),2), 'r*');
        end
        if (i<frame_number)
           M(i) = getframe;
        end
    %from green to red LED measured from normal x-axis on cartesian
    %coordinates
    si = atan2(centroids_R(robots_R(1),2)-(centroids_L(robots_L(1),2))...
        ,(centroids_L(robots_L(1),1))-centroids_R(robots_R(1),1))*180/pi;

    %makes sure angle is from 0 to 360 degrees
     if (si < 0)
         si = si + 360;
```

```
    end
    %converts to radians
    si = si*pi/180;

    error_si = sid-si;
    if (error_si > pi)
        error_si = error_si-(360*pi/180);
    end
    if (error_si < -pi)
            error_si = error_si+(360*pi/180);
    end


    pos = [1/2*(centroids_R(robots_R(1),1)-
centroids_L(robots_L(1),1))+centroids_L(robots_L(1),1)...
        , 1/2*(centroids_R(robots_R(1),2)-
centroids_L(robots_L(1),2))+centroids_L(robots_L(1),2)];
    x_pose = pos(1);
    y_pose = 480-pos(2);

    %Logs x,y,si,and time
    x_log(i)=x_pose;
    y_log(i)=y_pose;
    si_log(i) = si;
    timer = toc;
    time_log(i) = timer;

   end
   end
```

```
%++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
%++++++++++++++++++++++START CONTROL++++++++++++++++++++++++++++++++++++++++
%++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%DEFINE MEMBER MATRICIES%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Rot = [cos(si) -sin(si) 0 ;
       sin(si)  cos(si) 0 ;
          0         0   1];

%converts pixels  to m
x = x_pose*x_conversion;
y = y_pose*y_conversion;


% vectors
error = [xd-x yd-y error_si]';
```

```
Pd = [xd yd sid]';
P1 = [x y si]';
%=========================================
%CALCULATE ERROR DERIVATIVE
%=========================================

% derivatives using rise over run technique and integrals
% using the trapazoidal rule
% B = [-1 0 0;
%      0 -1 -1;
%      0 -r2*cos(theta2) 0];
if(i==1)
theta_hat = [1.5 0 0 0 .5 .7 0 .3 0]';
end

B_hat = [b11_sgn*theta_hat(1) b12_sgn*theta_hat(2) b13_sgn*theta_hat(3);
         b21_sgn*theta_hat(4) b22_sgn*theta_hat(5) b23_sgn*theta_hat(6);
         b31_sgn*theta_hat(7) b32_sgn*theta_hat(8) b33_sgn*theta_hat(9)];

Y_adp = [b11_sgn*(cos(si)*Ua_age)/m, b12_sgn*(cos(si)*Ub_age)/m,
b13_sgn*(cos(si)*Uc_age)/m,...
    b21_sgn*(sin(si)*Ua_age)/m, b22_sgn*(sin(si)*Ub_age)/m,
b23_sgn*(sin(si)*Uc_age)/m, 0, 0, 0;
    b11_sgn*-(sin(si)*Ua_age)/m, b12_sgn*-(sin(si)*Ub_age)/m, b13_sgn*-
(sin(si)*Uc_age)/m,...
    b21_sgn*(cos(si)*Ua_age)/m, b22_sgn*(cos(si)*Ub_age)/m,
b23_sgn*(cos(si)*Uc_age)/m, 0, 0, 0;
    0, 0, 0, 0, 0, 0, b31_sgn*Ua_age/j, b32_sgn*Ub_age/j, b33_sgn*Uc_age/j];

if (i==1)
    error_dot = [0 0 0]';
    Pd_dot = [0 0 0]';
    P1_dot = [0 0 0]';
    Pd_dot_dot = [0 0 0]';
else
    error_dot = (error-error_age)/(timer-time_age);
    Pd_dot = (Pd-Pd_age)/(timer-time_age);
    P1_dot = (P1-P1_age)/(timer-time_age);
    Pd_dot_dot = (Pd_dot-Pd_dot_age)/(timer-time_age);
end



%error signal r
r_error = error_dot + k_alpha * error;
%%%%%%%need to set up integrator
theta_hat_dot = 1.0*-Y_adp'*r_error;

% theta_hat_dot(6) = 10*theta_hat_dot(6);

if (theta_hat(1) <= theta_hat_min && theta_hat_dot(1)<0.0)
    theta_hat_dot(1) = 0.0;
end

if (theta_hat(5) <= theta_hat_min && theta_hat_dot(5)<0.0)
```

```
    theta_hat_dot(5) = 0.0;
end

if (theta_hat(6) <= theta_hat_min && theta_hat_dot(6)<0.0)
    theta_hat_dot(6) = 0.0;
end

if (theta_hat(8) <= theta_hat_min && theta_hat_dot(8)<0.0)
    theta_hat_dot(8) = 0.0;
end



if(i~=1)
theta_hat = theta_hat+((timer-time_age)*(theta_hat_dot +
theta_hat_dot_age)/2);
end

if(theta_hat(1)<= theta_hat_min)
    theta_hat(1) = theta_hat_min;
end

if(theta_hat(5)<= theta_hat_min)
    theta_hat(5) = theta_hat_min;
end

if(theta_hat(6)<= theta_hat_min)
    theta_hat(6) = theta_hat_min;
end

if(theta_hat(8)<= theta_hat_min)
    theta_hat(8) = theta_hat_min;
end


%//================================================
%//implement full controller
%...
%//================================================
U = inv(Rot*inv(Mass)*B_hat)*((Pd_dot_dot+k_alpha*error_dot)...
+(P1_dot(3)*s1*P1_dot)+(Kr*r_error)+(Rot*inv(Mass)*Drag*Rot'*P1_dot));
%Allocate thrusts
Ua = U(1);
Ub = U(2);
Uc = U(3);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%AGE THE LOCATION TERMS%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Pd_age = Pd;
P1_age = P1;
Pd_dot_age = Pd_dot;
error_age = error;
time_age = timer;
theta_hat_dot_age = theta_hat_dot;
```

```
Ua_age = Ua;
Ub_age = Ub;
Uc_age = Uc;
%++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
%++++++++++++++++++++++END CONTROL++++++++++++++++++++++++++++++++++++++++++
%++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%SEND THRUSTS TO THE BOAT%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        if (communication ==1)

              my_string1 = sprintf('a%5.3f\r', Ua);
              fprintf(s3, my_string1);

              my_string2 = sprintf('b%5.3f\r', Ub);
              fprintf(s3, my_string2);

              my_string3 = sprintf('c%5.3f\r', Uc);
              fprintf(s3, my_string3);

        end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%CALCULATE AND LOG INDIVIDUAL THRUSTS%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

            U1 = 0.5*(Ua+sqrt(Ua*Ua+gam0*gam0));
            U4 = abs(0.5*(-Ua+sqrt(Ua*Ua+gam0*gam0)));

            U2 = 0.5*(Ub+sqrt(Ub*Ub+gam0*gam0));
            U5 = abs(0.5*(-Ub+sqrt(Ub*Ub+gam0*gam0)));

            U3 = 0.5*(Uc+sqrt(Uc*Uc+gam0*gam0));
            U6 = abs(0.5*(-Uc+sqrt(Uc*Uc+gam0*gam0)));

            if (U1>2.4)
                U1 = 2.4;
            end
            if (U4>2.4)
                U4 = 2.4;
            end
            if (U2>1.6)
                U2 = 1.6;
            end
            if (U3>1.6)
                U3 = 1.6;
            end
            if (U5>1.6)
                U5 = 1.6;
            end
            if (U6>1.6)
                U6 = 1.6;
            end
```

```
            U1_log(i) = U1;
            U4_log(i) = U4;
            U2_log(i) = U2;
            U5_log(i) = U5;
            U3_log(i) = U3;
            U6_log(i) = U6;

            B_hat_log(1,i) = B_hat(1,1);
            B_hat_log(2,i) = B_hat(1,2);
            B_hat_log(3,i) = B_hat(1,3);
            B_hat_log(4,i) = B_hat(2,1);
            B_hat_log(5,i) = B_hat(2,2);
            B_hat_log(6,i) = B_hat(2,3);
            B_hat_log(7,i) = B_hat(3,1);
            B_hat_log(8,i) = B_hat(3,2);
            B_hat_log(9,i) = B_hat(3,3);

    end



if(i>frame_number)
   flag = 1;
end
i=i+1;


end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%RESET VARIABLES AND PROPERLY TERMINATE PROGRAM%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
flag = 0;
avg_time = mean(frame_times(3:end) - frame_times(2:end-1))
%close all
%plot(x_m_p*pose_hist(:,1), y_m_p*po se_hist(:,2),'*-')
%grid on;

% This is frame time
%plot(frame_times(3:end) - frame_times(2:end-1))


delete(vidobj);
if (communication ==1)
fclose(s3);
delete(s3);
end
```

16.4 Performance Index

Performance_index_rev:

```
%Calculates the error index, thrust index, and settling time
run_it = 151;
x_log_pt = x_log*7.927/640;
y_log_pt = y_log*4.4/480;
Pm = 0.0;
Ti = 0.0;
flag = 0;
stopflag = 0;

for i = 1:run_it
    %construct the error vector
    error(:,i) = Pd-[x_log_pt(i) y_log_pt(i) si_log(i)]';
    error(3,i) = error(3,i)*0.46;

    %calculate the error index
    Pm = Pm + norm(error(:,i));
    %calculate the thrust index
    Ti = Ti + U1_log(i)+U2_log(i)+U3_log(i)+U4_log(i)+U5_log(i)+U6_log(i);
    %calculate the settling time
    if (norm(error(:,i)) <= 0.44 && stopflag==0)
        flag = flag+1;
        if (flag>5)
            t_settle = time_log(i)
            stopflag = 1;
        end
    end


end
%normalize the error and thrust indicies
Performance = Pm/run_it
Thrust_avg = Ti/run_it
```

16.5 Vessel's C-code

Gain_control_remote.c:

```
// ===========================================================================
// Name                 : Erik T Smith
// Description  : Tugboat Position control, Closed-Loop, Position and
//                              Heading Control
// Date                 : 08-FEB-2007
// Slave program to Matlab, outputs counts when given thrusts
// ===========================================================================




// ================================================
// Rabbit Parameters
// ================================================
#define BOUTBUFSIZE 127
#define BINBUFSIZE 127
#define EOUTBUFSIZE 127
#define EINBUFSIZE 127
```

```
#define COUTBUFSIZE 127
#define CINBUFSIZE 127


// ================================================
// Sensor Coefficients
// ================================================
#define pi              (3.1415926)
// pi


// ================================================
// Prototypes
// ================================================

xmem nodebug void MsDelay(int);

xmem nodebug void move_servos(int,int,int,int,int,int);

// ================================================
// Declare Global Variables
// ================================================



//
================================================================================
=====================================
// Main()
//
================================================================================
=====================================
void main(void)
{
    // ================================================
    // Local variable declarations
    // ================================================
    int s1, s2, s3, s4, s5, s6;
    char sentence[20];
    char inchar[12];
    char outchar[6];
    float enable;
    char recCmd;
    char c;
    int z, i;
    long count;
    float Ua, Ub, Uc, U1, U2, U3, U4, U5, U6, gam0, exit, e_time;



    // ================================================
    // Initialize Rabbit SBC
    // ================================================
```

```
    //--------------set baud rate for each serial port-----------------------
------------
    serEopen(9600);
    serBopen(19200);
    serCopen(9600);

    serEwrFlush();
    serErdFlush();
    serBwrFlush();
    serBrdFlush();
    serCwrFlush();
    serCrdFlush();

    //-------------turn off all motors that may be running-------------------
------------
    move_servos(128, 128, 128, 128, 128, 128);




    //-----------initialize variables---------------------------------------
------------
    i=0;
    c = 255;
    for(i=0;i<=11;i++)
    inchar[i]='\0';

    for(i=0;i<=5;i++)
    outchar[i]='\0';



    i = 0;
    z = 0;

    exit = 0.0;
    e_time = 105000.0;  //repetitions the program waits before shutting the
boat
    //down when no update is rec dfreeived


    gam0 = 0;
    Ua = 0.0;
    Ub = 0.0;
    Uc = 0.0;



    // ================================================
    // Main loop
    // ================================================
```

```
    while(1)
    {

    // ================================================
    // Inputs - measurements
    // ================================================
        recCmd = 'N';  //RECCMD FLAG IS RESET TO NO

        while((recCmd == 'N') && (exit < e_time))
        {
        //---------------------RECEIVE COMMAND LOOP-------------------------
----------------
        //waits until it receives a character other than ascii 255

        while(exit<e_time)          // loop until we get a character
            {
                //reads in a character from serial port B

                c = serBgetc();
            //if character is different than ascii 255 then exit loop
            if (c != 255)
            {
                break;
                exit = 0.0;
            }
            exit++;
            }
            //---------------------CHARACTER CHECK-------------------------
--------------------
            //if a character is \r then the string is received, else put the
character in inchar

            if(c == 13)
            {
                recCmd = 'Y';
                inchar[i] = '\0';
                i=0;
                }
                else if(c !=10)  //Matlab appends a line feed (dec 10) to
each output so must not input
                {
                    inchar[i] = (c & 0x7F); //forces 7 bit ascii
                    // printf("%c",c);
                    i++;
                }

                //initialize c to ascii 255
                c = 255;
        }
//=============================================================================
==============================================
//For the following if statements, the 2nd through 4th elements must be
digits for the statement to get converted and assigned
//=============================================================================
==============================================
```

```
    if(inchar[0] == 97)
    //if 1st character in string is an 'a' then inchar is saved as Ua
        {
            for (z=0;z<=4;z++)
                outchar[z] = inchar[z+1];
            Ua = atof(outchar);
            printf("Ua = %f\n",Ua);
            count++;
        }

    if(inchar[0] == 98)
    //if 1st character in string is an 'b' then inchar is saved as Ub
        {
            for (z=0;z<=4;z++)
                outchar[z] = inchar[z+1];
            Ub = atof(outchar);
            printf("Ub = %f\n",Ub);
            count++;
        }

    if(inchar[0] == 99)
    //if 1st character in string is an 'c' then inchar is saved as Uc
        {
            for (z=0;z<=4;z++)
                outchar[z] = inchar[z+1];
            Uc = atof(outchar);
            printf("Uc = %f\n",Uc);
            count++;
        }


// ================================================
// Clear variables
// ================================================


for(i=0;i<=5;i++)
outchar[i]='\0';
for(i=0;i<=11;i++)
inchar[i]='\0';
i=0;

// ================================================
// Calculate counts
// ================================================

    if(fmod((float)count,3.0) == 0.0 && count!=0 && exit < e_time)
    {

        count = 0;

        //=========================================
        //each thrust output in (N)
        //=========================================
```

```
            U1 = 0.5*(Ua+sqrt(Ua*Ua+gam0*gam0));
            U4 = fabs(0.5*(-Ua+sqrt(Ua*Ua+gam0*gam0)));

            U2 = 0.5*(Ub+sqrt(Ub*Ub+gam0*gam0));
            U5 = fabs(0.5*(-Ub+sqrt(Ub*Ub+gam0*gam0)));

            U3 = 0.5*(Uc+sqrt(Uc*Uc+gam0*gam0));
            U6 = fabs(0.5*(-Uc+sqrt(Uc*Uc+gam0*gam0)));

            if (U1>2.4)
                U1 = 2.4;
            if (U4>2.4)
                U4 = 2.4;
            if (U2>1.6)
                U2 = 1.6;
            if (U3>1.6)
                U3 = 1.6;
            if (U5>1.6)
                U5 = 1.6;
            if (U6>1.6)
                U6 = 1.6;


            // printf("U1=%f U2=%f U3=%f U4=%f U5=%f
U6=%f\r\n",U1,U2,U3,U4,U5,U6);

            //=========================================
            //Calculate counts for each thruster
            //=========================================

            //counts for all 1000gph thrusters 1,4
            //3rd order curve fit using a_hat = 109.2619, b_hat = -72.8738,
c_hat = 64.8534, d_hat = -22.3762
            s1= (int)(109.2619 + -72.8738*U1 + 64.8534*U1*U1 + -
22.3762*U1*U1*U1);
            s4= (int)(109.2619 + -72.8738*U4 + 64.8534*U4*U4 + -
22.3762*U4*U4*U4);


            //counts for all 800gph thrusters 2,3,5,6
            //3rd order curve fit using a_hat = 108.1623, b_hat = -57.3317,
c_hat = 46.4793, d_hat = -31.0171
            s2= (int)(108.1623 + -57.3317*U2 + 46.4793*U2*U2 + -
31.0171*U2*U2*U2);
            s3= (int)(108.1623 + -57.3317*U3 + 46.4793*U3*U3 + -
31.0171*U3*U3*U3);
            s5= (int)(108.1623 + -57.3317*U5 + 46.4793*U5*U5 + -
31.0171*U5*U5*U5);
            s6= (int)(108.1623 + -57.3317*U6 + 46.4793*U6*U6 + -
31.0171*U6*U6*U6);

            //=====================================================
            //force all counts greater than 1
```

```
            if (s1<1)
                s1 = 1;
            if (s2<1)
                s2 = 1;
            if (s3<1)
                s3 = 1;
            if (s4<1)
                s4 = 1;
            if (s5<1)
                s5 = 1;
            if (s6<1)
                s6 = 1;

            move_servos(s1, s2, s3, s4, s5, s6);
        }

        if (exit >= e_time)
        {
            move_servos(128, 128, 128, 128, 128, 128);
            exit = 0.0;
        }


   }
}



//
================================================================================
=====================================
// Library Functions
//
================================================================================
=====================================


////////////////////////////////////////////////////////////////////

/*** Beginheader MsDelay */
void MsDelay(int MS);
/*** endheader */

/* START FUNCTION DESCRIPTION ****************************************
MSDelay                 <ES308_SBC.LIB>

SYNTAX:        MsDelay(int MS);


DESCRIPTION:    This function causes processing to be delayed for the
specified
                    number of milliseconds.  It should not be used within a
costatement.
```

```
                    (Use waitfor(DelayMS(ms)) in costatements.

                    See also SecDelay(sec).

RETURN VALUE:   None

END DESCRIPTION **********************************************************/

xmem void MsDelay(int MS)   // Millisecond delay
   {
    // 12/13/02 Modified with code below V2.7 - wml

    auto unsigned long   done_time;
    done_time = MS_TIMER + MS;
    while((long)(MS_TIMER - done_time) <0) {/* do nothing */}

    // Versions  before V2.7
    //long SavTimer, TimerDiff;
    //TimerDiff = 0;
    //SavTimer = MS_TIMER;
    //while(TimerDiff < MS) {TimerDiff = MS_TIMER - SavTimer;}
   }
#memmap xmem



xmem nodebug void move_servos(int s1, int s2, int s3,int s4, int s5, int s6)
//SV203 Board
{
    char temp[10];

    serEputs("BD1");  //initializes board # 1
    serEputs("SV1");  //sends count number to servo 1
    serEputs("M");
    itoa(s1, temp);
    serEputs(temp);
    temp[0] = '\0';

    serEputs("BD1");  //initializes board # 1
    serEputs("SV2");  //sends count number to servo 2
    serEputs("M");
    itoa(s2, temp);
    serEputs(temp);
    temp[0] = '\0';

   serEputs("BD1");  //initializes board # 1
    serEputs("SV3");  //sends count number to servo 3
    serEputs("M");
    itoa(s3, temp);
    serEputs(temp);
    temp[0] = '\0';
```

```
    serEputs("BD1");  //initializes board # 1
    serEputs("SV4");  //sends count number to servo 4
    serEputs("M");
    itoa(s4, temp);
    serEputs(temp);
    temp[0] = '\0';

    serEputs("BD1");  //initializes board # 1
    serEputs("SV5");  //sends count number to servo 5
    serEputs("M");
    itoa(s5, temp);
    serEputs(temp);
    temp[0] = '\0';

   serEputs("BD1");  //initializes board # 1
    serEputs("SV6");  //sends count number to servo 6
    serEputs("M");
    itoa(s6, temp);
    serEputs(temp);
    temp[0] = '\0';

    serEputc('\r'); //needs this code to process the command
    temp[0] = '\0';
}
#memmap xmem
```